

# Use of Internet Embedding Tools for Heterogeneous Resources Aggregation

Olivier Beaumont  
INRIA Bordeaux Sud-Ouest  
Bordeaux, France  
Email: [olivier.beaumont@labri.fr](mailto:olivier.beaumont@labri.fr)

Nicolas Bonichon  
Université de Bordeaux  
Bordeaux, France  
Email: [nicolas.bonichon@labri.fr](mailto:nicolas.bonichon@labri.fr)

Philippe Duchon  
Université de Bordeaux  
Bordeaux, France  
Email: [philippe.duchon@labri.fr](mailto:philippe.duchon@labri.fr)

Hubert Larchevêque  
Institut Polytechnique de Bordeaux  
Bordeaux, France  
Email: [hubert.larcheveque@labri.fr](mailto:hubert.larcheveque@labri.fr)

**Abstract**—In this paper we are interested in large scale distributed platforms like BOINC, consisting of heterogeneous resources and using Internet as underlying communication network. In this context, we study a resource clustering problem, where the goal is to build clusters having at least a given capacity and such that any two participants to the same cluster are not too far from each other. In this context, the distance between two participants corresponds to the latency of a communication between them. Our goal is to provide algorithms with provable approximation ratios. In such large scale networks, it is not realistic to assume that the whole latency matrix (that gives the latency between any two participants) is known, and we need to rely on embedding tools such as Vivaldi or Sequoia. These tools enable to work on compact descriptions and well described metric spaces in which the distance between two points can be obtained directly from a small amount of information available at each node. We present the Bin Covering under Distance Constraint problem (BCDC for short), and propose dedicated algorithms for this problem for each metric space induced by each of the embedding tools. Then, we propose a comparison of these algorithms based on actual latency measures, that enables to decide which algorithm/embedding tool pair offers in practice for realistic datasets the best balancing between distance prediction and approximation ratios for the resource clustering problem.

**Keywords**—Internet embedding, resource clustering

## I. INTRODUCTION

In this paper, we study the use of two embedding tools for Internet resource clustering in large scale distributed platforms. Internet embedding tools are used to map distributed resources connected through Internet into a simple (usually metric) space. Among the most widely encountered embedding tools in the literature are Vivaldi [1], [2] and Sequoia [3], [4]. These tools assign to each resource a position in a simple space, such that the distance (the latency) between any two nodes can be approximated by their distance in the simple space. Of course, another possibility would consist in computing and using the latency matrix  $\mathcal{L}$ , where  $\mathcal{L}_{i,j}$  denotes the measured latency between resources  $i$  and  $j$ . Nevertheless, this approach has two main

drawbacks. First, in the context of large scale distributed networks, it is unrealistic to assume that all  $\mathcal{L}_{i,j}$  values can be determined accurately due to the cost to perform all measures. Second, when using the latency matrix as input, we work in the most general space without any specific topological property. Designing efficient (with good approximation ratios) algorithms in this context turns out to be extremely difficult. On the other hand, embedding tools induce a (small) distortion of latencies, but they enable to work in simpler spaces and therefore to design efficient approximation algorithms. In the context of a given application, such as resource clustering, only the performance of the pair embedding/clustering algorithm is meaningful. In this paper, we consider 3 embeddings : no embedding (direct use of the latency matrix), Vivaldi [1], [2] and Sequoia [3], [4]. For each embedding, we design a specific clustering algorithm using the structural properties of the resulting space, and we compare the performance of the embedding/algorithm pair using datasets based on PlanetLab (<http://www.planet-lab.org/>) latency measurements.

Since Internet latencies space contains a lot of triangular inequality violations [5], it can be described as a semi-metric space (a space where the distance function does not satisfy the triangular inequality). Thus, it cannot be embedded in a metric space without encountering a loss of accuracy. In fact, Internet latencies space seems to be close to an infra-metric [6], a space in which the triangular inequality is relaxed in the following way :  $d(u,v) \leq \rho \cdot \max(d(u,w), d(v,w))$  for any triple of nodes  $(u,v,w)$ . Results presented in [6] show that most triples satisfy above inequality with  $\rho = 2$ , and almost every triples satisfy it with  $\rho = 10$ . In this paper, we will consider semi-metric spaces in which most of triples satisfy this inequality for a small value of  $\rho$  and approximation results will be given as a function of  $\rho$ .

Bin covering [7] is a classical problem that has been studied under many variants. In this paper, we study the case where we are given a set of weighted (heterogeneous)

elements (resources)  $S$  with distances (latencies) between each pair of elements. In this paper, we consider the generalization where a distance constraint is added. More specifically, we consider that the distance between any two elements belonging to the same bin (cluster) is lower than a given threshold  $d_{\max}$ . Theoretically, in this generalization, elements belong to a metric space, to an infra-metric space, or even to a space in which distances between nodes are given by a latency matrix (and therefore not necessarily satisfying any kind of triangular inequality). In BCDC, the goal is to build a maximum of bins of diameter no more than  $d_{\max}$ , weighting at least  $W$ , a given threshold.

The motivation for studying BCDC can be found in details in [8], [9], where this problem was first studied. It is related to highly distributed platforms, such as BOINC [10] or Folding@home<sup>1</sup>. Until now, all the applications running on these platforms consist of a huge number of independent tasks, and all data necessary to process a task must be stored locally at the processing node. We consider the case where the data set needed to perform a task is possibly too large to be stored in a single node. Then, both processing and storage must be distributed on a small set of nodes that will collaborate to perform the task. The nodes involved in a given cluster should have an aggregate capacity (memory, processing power...) higher than a given threshold, and they should be close enough (the latencies between any two nodes belonging to the same cluster should be small enough) in order to avoid high communication latencies. Building such clusters is equivalent to solving BCDC, where the distance between two nodes corresponds to the latency between these nodes.

In this paper we present approximation algorithms based on resource augmentation techniques [11], [12]. We compare the number of bins built by a polynomial time algorithm allowed to build bins of diameter at most  $\beta d_{\max}$ , for  $\beta > 1$ , to the optimal number of bins of diameter at most  $d_{\max}$ .

More precisely, we will say that  $\mathcal{A}$  is an  $(\alpha, \beta)$ -approximation algorithm if it runs in polynomial time, builds bins of diameter at most  $\beta d_{\max}$ , and if the number of built bins is at least  $\alpha OPT^*$ , where  $OPT^*$  is the optimal number of bins with the distance constraint set to  $d_{\max}$ .

In the context of large scale distributed platforms, resource augmentation is both efficient and realistic. Indeed, the aggregated power (or memory) of a cluster really needs to be larger than a given threshold in order to be able to process a task, whereas the threshold on the maximal latency between two nodes belonging to the same cluster is somehow weaker, since a cluster would still be able to work if it is violated (it would then simply work slowly).

In this context, if  $\beta = (2 - \epsilon)$ , BCDC is hard to approximate in the general case, within any constant approximation ratio (even any function in the number of elements), by a

reduction to the MAXIMUM CLIQUE problem. On the other hand, a  $(\frac{1}{3}, 4)$ -approximation algorithm has been presented in [9] for BCDC in  $\mathbb{R}^D$ .

In this paper we present in Section II-B a  $(\frac{2}{5}, \rho^2)$ -approximation algorithm for BCDC in a  $\rho$ -inframetric. We also present, in Section II-C a  $(\frac{2}{5}, 2 + \frac{3\sqrt{3}}{2})$ -approximation algorithm for BCDC in a space generated by Vivaldi. In Section II-D, we present a  $(\frac{1}{3}, 2)$ -approximation algorithm for BCDC in tree metric spaces, that can be used in a space generated by Sequoia. Then in Section III we present our simulations and the way to compare the different embedding tools in the context of resource clustering.

## II. DEFINITION AND APPROXIMATION ALGORITHMS

### A. Notations and definitions

An instance  $\mathcal{I}$  of BCDC can be described as a 5-tuple  $\mathcal{I} = (S, d, w, W, d_{\max})$  where  $S$  is a set  $S = \{e_1, \dots, e_n\}$  of elements,  $(S, d)$  is a semi-metric space,  $w$  is a weight function,  $W$  is a weight threshold and  $d_{\max}$  is the distance threshold.

Throughout this paper, for the sake of simplicity, we set  $W = 1$  and we normalize the weights of the elements accordingly (*i.e.* divide them by  $W$ ). Remember that in our context of Internet resource clustering, the weight of the elements correspond to the heterogeneous capacity (that may represent either processing, memory or storage) of participating resources. Therefore, this normalization corresponds to express all capacities as a percentage of the overall target capacity of the cluster. Moreover, we do not deal with elements whose weight is larger than 1. Indeed, such elements can form bins (clusters) by themselves and can be removed from  $\mathcal{I}$ . Thus, an instance of BCDC can be described by a 4-tuple  $\mathcal{I} = (S, d, w, d_{\max})$ , where  $w : S \rightarrow [0; 1[$ , hence the following definition.

*Definition 2.1 (BCDC):* Given an instance  $\mathcal{I} = (S, d, w, d_{\max})$ , find a collection of pairwise disjoint subsets  $S_1, \dots, S_K$  of  $S$  of maximal cardinality  $K$  such that  $\forall i \leq K, \sum_{e \in S_i} w(e) \geq 1, \forall (e_u, e_v) \in S_i, d(e_u, e_v) \leq d_{\max}$ .

To work on graphs, we need the following tool to build a graph from a set of points.

*Definition 2.2 (Compatibility Graph):* The compatibility graph  $\text{Comp}(\mathcal{I}, d)$  associated to an instance  $\mathcal{I}$  is the graph  $G = (S; E)$  such that  $\forall (e_i, e_j) \in S^2, (e_i, e_j) \in E \Leftrightarrow d(e_i, e_j) \leq d$ .

Observe that  $(e_i, e_j) \in E$  and  $(e_j, e_k) \in E \Rightarrow d(e_i, e_k) \leq \rho d$  in a  $\rho$ -inframetric space ( $\rho \leq 2$  in a metric space). Note that if  $S$  are points in a Euclidean space, and  $L_2$  norm is used to define distances,  $\text{Comp}(\mathcal{I}, 1)$  is the unit-disk graph [13].

### B. A Greedy $(\frac{2}{5}, \rho^2)$ -approximation algorithm in a $\rho$ -inframetric

In [9], a polynomial time  $(\frac{1}{3}, 4)$ -approximation algorithm for BCDC has been proposed. This algorithm is quite simple

<sup>1</sup><http://folding.stanford.edu>

to implement in a distributed way, and works in two phases. In this section, we present Algorithm 1, which is an improvement of the previous algorithm. Concerning the number of built bins, this algorithm ensures an approximation ratio of  $\frac{2}{5}$  in any semi-metric space. When this semi-metric space is a  $\rho$ -inframetric, Algorithm 1 is a  $(\frac{2}{5}, \rho^2)$ -approximation algorithm for BCDC. When the space is metric, it is a  $(\frac{2}{5}, 4)$ -approximation algorithm for BCDC. Otherwise, nothing is guaranteed on the diameter of the bins (on the other hand, as already noted, the values of  $\rho$  that are observed in practice are reasonably small).

To build bins, the algorithm is based on a classical strategy for Bin Covering, namely Next-Fit-Decreasing: it consists in sorting the elements in the non-increasing order of their weight. Then elements are considered in this order and put in the current bin, until the weight of this bin overtakes 1. When a weight larger than 1 is reached, the current bin is closed and a new one is opened. Without the distance constraint, this algorithm provides a  $\frac{1}{2}$  approximation ratio, and build bins of weight  $1 + b$  ( $b < 1$ ) containing only elements of weight strictly greater than  $b$ .

---

**Algorithm 1** Greedy  $(\frac{2}{5}, \rho^2)$ -approximation algorithm for BCDC

---

- 1:  $U \leftarrow S$  // Elements that do not belong to any bin
- 2:  $G = \text{Comp}(\mathcal{I}, d_{\max})$

Phase 1:

- 1: **for** each node  $e$  in  $S$  **do**
- 2:  $C = U \cap \{e \cup N(e)\}$
- 3: apply Next-Fit-Decreasing algorithm on  $C$
- 4: remove elements corresponding to each bin built from  $U$ .
- 5: **end for**

Phase 2:

- 1: apply Phase 1 on  $G = \text{Comp}(\mathcal{I}, \rho d_{\max})$ .
- 

*Theorem 2.1:* Algorithm 1 is a  $(\frac{2}{5}, \rho^2)$ -approximation algorithm for BCDC when  $d$  is a  $\rho$ -inframetric.

*Proof of Theorem 2.1:*

Clearly, all bins returned by Algorithm 1 have diameter at most  $\rho^2 d_{\max}$ , hence the resource augmentation ratio  $\rho^2$ . Indeed, all the elements belonging to a bin built during Phase 1 are at distance at most  $d_{\max}$  from the central node considered when this bin has been built, and thus two elements of such a bin are at distance at most  $\rho d_{\max}$ . Then, similarly, all the elements belonging to a bin built during Phase 2 are at distance at most  $\rho d_{\max}$  from the central node, and thus two elements of such a bin are at distance at most  $\rho^2 d_{\max}$ .

Let OPT be an optimal and thrifty solution, *i.e.* one for which in each bin of weight  $1 + w$ , elements have weight larger than  $w$ . Such a solution always exists since we can modify any optimal solution to enforce this property by

iteratively removing elements whose weight is smaller than  $w$ . Let us denote by  $\mathcal{B}^{(k)}$ ,  $k = 1, 2$  the set of bins built during Phase  $k$  and let  $b_k = |\mathcal{B}^{(k)}|$  and let us define the *extended area* of a bin  $B \in \mathcal{B}^{(1)}$  as the set of elements  $e \in S$  such that  $d(e; B) \leq d_{\max}$ , where  $d(e; B) = \min_{u \in B} d(e; u)$ .

First, let us prove that any bin  $B_{\text{OPT}}$  of OPT intersects at least one bin of  $\mathcal{B}^{(1)}$ . Indeed, Phase 1 ends when no bin of diameter lower than  $d_{\max}$  can be build, what means that at least one element of  $B_{\text{OPT}}$  has been added to a bin  $B$  of  $\mathcal{B}^{(1)}$ . Therefore, we can notice that  $B_{\text{OPT}}$  is included in the extended area of  $B$  since all its elements are at distance at most  $d_{\max}$  from  $B$ .

Let us partition the bins  $B_i^{(1)}$ ,  $1 \leq i \leq b_1$  of  $\mathcal{B}^{(1)}$  into two subsets  $\mathcal{K}_1$  and  $\mathcal{M}_1$ , depending on the number of elements  $|B_i^{(1)}|$  they contain:

$$\left\{ \begin{array}{l} \mathcal{K}_1 = \{B_i^{(1)} \in \mathcal{B}^{(1)}, |B_i^{(1)}| = 2\}, \quad k_1 = |\mathcal{K}_1| \\ \text{and let } \text{OPT}_K \text{ denote the set of bins of OPT that} \\ \text{intersect bins in } \mathcal{K}_1, \\ \mathcal{M}_1 = \mathcal{B}^{(1)} \setminus \mathcal{K}_1, \quad m_1 = b_1 - k_1 \text{ and let } \text{OPT}_M \text{ denote} \\ \text{the set of bins of } \text{OPT} \setminus \text{OPT}_K \text{ that intersect bins in } \mathcal{M}_1. \end{array} \right.$$

Recall that no bin can contain only one element since we do not consider elements whose weight is larger than 1 (see Section II-A). Since all bins of OPT intersect a bin built during Phase 1, then  $|\text{OPT}| \leq |\text{OPT}_K| + |\text{OPT}_M|$ .

Since each bin in  $\mathcal{K}_1$  consists of exactly 2 elements, and since bins created by OPT are disjoint, then each bin of  $\mathcal{K}_1$  intersects at most 2 bins of  $\text{OPT}_K$ . Hence,  $|\text{OPT}_K| \leq 2k_1$ .

In order to bound the cardinality of  $\text{OPT}_M$ , let us introduce  $w_{\text{lost}}$ , the overall weight of elements that do not belong to any bin returned by Algorithm 1, but that belong to  $\text{OPT}_M$ .

*Lemma 2.2:*  $w_{\text{lost}} < m_1$

*Proof:* In the extended area of any bin belonging to  $\mathcal{M}_1$ , the overall weight of elements that do not belong to a bin returned by Algorithm 1 is strictly less than 1. Indeed, if it was not the case, another bin would have been built during Phase 2. Therefore,  $w_{\text{lost}} < m_1$ . ■

Furthermore,  $|\text{OPT}_M| \leq w(\text{OPT}_M) \leq w(\mathcal{M}) + w(\mathcal{B}^{(2)}) + w_{\text{lost}}$

*Lemma 2.3:* If  $B \in \mathcal{M}_1$ , then  $w(B) < 1 + \frac{1}{|B|-1}$

*Proof:* Let  $w(B) = 1 + w$ . In Algorithm 1, the Next Fit Decreasing strategy is used to build bins, and as soon as the weight of  $B$  is larger than 1,  $B$  is closed and no more element is added to it. Hence the last element added to  $B$  is weighting more than  $w$ . Since elements are added in the decreasing order of their weight, any element of  $B$  weights at least  $w$ . Then,  $w(B) > w \times |B|$  and thus  $1 + w > w \times |B|$ . Therefore,  $w < \frac{1}{|B|-1}$ . ■

Thus, since every bin  $B$  of  $\mathcal{M}_1$  contains at least 3 elements, then  $w(B) \leq \frac{3}{2}$ . Consequently,  $w(\mathcal{M}_1) < \frac{3}{2}m_1$ . Since each bin of  $\mathcal{B}_2$  has weight at most 2, then

$$\begin{aligned} |\text{OPT}_M| &< \frac{3}{2}m_1 + 2b_2 + m_1 = \frac{5}{2}m_1 + 2b_2 \\ |\text{OPT}| &< \frac{5}{2}m_1 + 2b_2 + 2k_1 < \frac{5}{2}(b_1 + b_2), \end{aligned}$$

which concludes the proof of Theorem 2.1. ■

C.  $(\frac{2}{5}, 2 + \frac{3\sqrt{3}}{2})$ -approximation algorithm for BCDC in a space generated by Vivaldi

We present Algorithm 2, that is an adaptation of Algorithm 1 (designed for the general case of semi-metric spaces) to the case when elements are placed in a metric space generated by Vivaldi [1], [2]. Algorithm 2 builds bins of diameter  $\frac{\sqrt{3+1}}{2}d_{\max}$  in a first step, and  $\frac{\sqrt{3+1}}{2}(2 + \frac{\sqrt{3+1}}{2}) = 2 + \frac{3\sqrt{3}}{2}$  then (instead of  $2d_{\max}$  and  $4d_{\max}$  in the general case, in any metric space),

Vivaldi is one of the embedding tools we study in this paper. In Section III, it will be compared through experimental evaluation to different embedding tools. It associates to each node of a network two coordinates in the Euclidean plane, plus a height. In such a space we call Vivaldi space, the distance between two nodes having coordinates  $(a_x, a_y, a_h) \in \mathbb{R}^2 \times \mathbb{R}^+$  and  $(b_x, b_y, b_h) \in \mathbb{R}^2 \times \mathbb{R}^+$  is given by  $d(a, b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2} + a_h + b_h$ . This embedding allows good prediction of the latencies between each pair of nodes, as evaluated in [1], [2].

To describe Algorithm 2, we define the *lens* of two elements as follows.

**Definition 2.3 (Lens of  $e_i$  and  $e_j$ ):** Let  $e_i$  and  $e_j$  be two points. The (symmetric) lens of  $e_i$  and  $e_j$ , denoted by  $Le(e_i, e_j)$  is the set of points  $e_k$  such that  $d(e_i, e_k) \leq d(e_i, e_j)$  and  $d(e_j, e_k) \leq d(e_i, e_j)$ .  $d(e_i, e_j)$  is the diameter of the lens.

We also use a value  $\beta'$ , and prove in what follows that in a Vivaldi space,  $\beta' = \frac{\sqrt{3+1}}{2}$ .

---

**Algorithm 2**  $(\frac{2}{5}, \beta'(2 + \beta'))$ -approximation algorithm for BCDC.

---

- 1:  $U \leftarrow S$  // Elements not belonging to any bin
- 2:  $G = \text{Comp}(\mathcal{I}, \beta'd_{\max})$ .
- 3:  $d \leftarrow d_{\max}$

Phase 1:

- 1: **while** a lens of diameter at most  $d_{\max}$  contains a clique of  $G$  of weight greater than 1 **do**
- 2:   apply Next-Fit-Decreasing on such a clique
- 3:   remove elements corresponding to each bin built from  $U$ .
- 4: **end while**

Phase 2:

- 1:  $G = \text{Comp}(\mathcal{I}, \beta'(\beta' + 2)d_{\max})$ .
  - 2:  $d \leftarrow (\beta' + 2)d_{\max}$
  - 3: apply Phase 1.
- 

Since each clique of diameter  $d_{\max}$  is included in a lens of diameter at most  $d_{\max}$ , building cliques of diameter  $d_{\max}$  and weight larger than 1 simply consists in examining all such lenses (a lens is associated to a pair elements of  $S$

and therefore, there is a polynomial number of lenses), and identifying in each such lens valid bins to build.

Given two elements  $e_i, e_j \in S^2$ , in the lens  $Le(e_i, e_j)$  of diameter at most  $d_{\max}$  we define the graph  $G_{e_i, e_j} = \text{Comp}(\mathcal{I}, \beta'd_{\max}) \cap Le(e_i, e_j)$ . We prove in what follows, using Lemma 2.4, that  $\overline{G_{e_i, e_j}}$  is a bipartite graph, for  $\beta' = \frac{\sqrt{3+1}}{2}$ .

Therefore, identifying a valid bin in  $Le(e_i, e_j)$  corresponds to finding a weighted independent set of weight larger than 1 in  $\overline{G_{e_i, e_j}}$ . Finding a maximum weighted independent set in a bipartite graph can be done by adapting to a weighted version the classical solution for maximum independent set in bipartite graphs that, using König's Theorem [14], can be solved as a matching problem using Hall's Theorem. Therefore, if its weight is larger than 1, since it is a maximum weighted clique in  $\text{Comp}(\mathcal{I}, \frac{\sqrt{3+1}}{2}d_{\max})$ , a bin can be built.

**Lemma 2.4:** Given two points  $i, j$  in a Vivaldi space, such that  $i = (-a, 0, h_i)$  and  $j = (a, 0, h_j)$  for some  $a \geq 0$ , and given two points  $u = (x_u, y_u, h_u)$  and  $v = (x_v, y_v, h_v)$  in  $Le(i, j)$ , then if  $y_u y_v \geq 0$ ,  $d(u, v) \leq \frac{\sqrt{3+1}}{2}$ .

Note that the restriction  $y_i = y_j = 0$  implies no restriction on the points considered, but is just used to describe the lemma in a simple way.

Lemma 2.4 directly implies that  $\overline{G_{e_i, e_j}}$  is bipartite when  $\beta' = \frac{\sqrt{3+1}}{2}$ .

*Proof of Lemma 2.4:*

In what follows, each element  $e$  has three coordinates, denoted  $(x_e, y_e, h_e)$ . Given  $e = (x_e, y_e, h_e)$ , we denote by  $E$  its mapping on the Euclidean plane, with coordinates  $(x_e, y_e)$ .

Note that every lens  $Le(i, j)$  related to a pair  $i = (-a, 0, h_i)$  and  $j = (a, 0, h_j)$  with  $a \geq 0$ ,  $h_i \leq 1$  and  $h_j \leq 1$  is included in the lens  $Le(i', j')$  with  $i' = (-a + h_i, 0, 0)$  and  $j' = (a + h_j, 0, 0)$ .

Thus without loss of generality we study the lens  $Le(k, l)$ , with  $k = (-1/2, 0, 0)$  and  $l = (1/2, 0, 0)$  (see Figure 1). We also use the point  $s = (0, \frac{\sqrt{3}}{2}, 0)$  at the intersection of the two circles centered respectively in  $k$  and  $l$  and of radius  $d(k, l) = 1$ , the point  $t = (0, 0, 1/2)$ , and the point  $c = (0, \frac{\sqrt{3}-1}{4}, 0)$ , at the same distance of  $s$  as of  $t$ .

In the rest of the proof, we study the "right upper side" of the lens, *i.e.* the part of the lens containing points  $p$  such that  $x_p \geq 0$  and  $y_p \geq 0$ . We show that this right upper side of the lens is included in  $B(c, \frac{d(s, t)}{2})$ . Using a symmetry argument proves the lemma for the upper side of the lens. Using another symmetry argument ends the proof for the other side of the lens.

Let us first consider the point  $q = (1/2, \frac{\sqrt{3}}{2}, 0)$ , and let  $m = (x_m, y_m, h_m)$  with  $x_m \geq 0$ ,  $y_m \geq 0$  and  $h_m \geq 0$  be a point in the Vivaldi space included in the considered part of the lens.  $M$  belongs to the rectangle  $STQL$ .

W.l.o.g, suppose that  $d(k, m) = d(K, M) + h_m = 1$ .

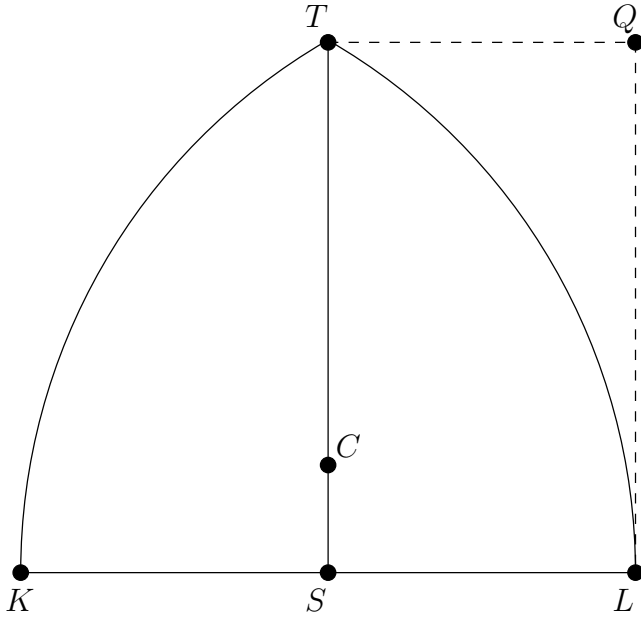


Figure 1. Illustration of notations used in the proof of Lemma 2.4

Consider the hyperbola branch of foci  $K$  and  $C$  in the euclidean plane of the projections, defined by the equation  $d(X, K) - d(X, C) = \frac{3-\sqrt{3}}{4}$ , for  $X$  in this euclidean plane.

We can notice that  $d(S, K) - d(S, C) = d(T, K) - d(T, C) = \frac{3-\sqrt{3}}{4}$ , thus  $S$  and  $T$  are exactly on the hyperbola.

Then,  $d(L, K) - d(L, C) = 1 - \sqrt{\frac{4-\sqrt{3}}{8}} \geq \frac{3-\sqrt{3}}{4}$  and  $d(Q, K) - d(Q, C) = \sqrt{\frac{7}{4}} - \sqrt{\frac{4+\sqrt{3}}{8}} \geq \frac{3-\sqrt{3}}{4}$ . Thus these two points are on the same side of the hyperbola. Since both this side of the hyperbola and the rectangle  $STQL$  are convex, then all the points  $M$  in the rectangle  $STQL$  satisfy  $d(M, K) - d(M, C) \geq \frac{3-\sqrt{3}}{4}$ .

Since  $d(k, m) = d(K, M) + h_m = 1$ , we have:

$$\begin{aligned} d(m, c) &= d(M, C) + h_m \\ d(m, c) &\leq d(M, K) - \frac{3-\sqrt{3}}{4} + h_m \\ d(m, c) &\leq 1 - \frac{3-\sqrt{3}}{4} \\ d(m, c) &\leq \frac{\sqrt{3}-1}{4}, \end{aligned}$$

which concludes the proof of Lemma 2.4. ■

The weight analysis concerning Algorithm 2 is exactly the same as for Theorem 2.1. The required resource augmentation corresponds to the largest diameter a bin can have, hence  $2 + \frac{3\sqrt{3}}{2}d_{\max}$  in this case.

**Theorem 2.5:** Algorithm 2 is a  $(\frac{2}{5}, 2 + \frac{3\sqrt{3}}{2})$ -approximation algorithm for BCDC in a Vivaldi space.

In fact, the above algorithm works in any metric space where the existence of a clique of sufficient weight can be detected in polynomial time. In a Vivaldi space, this detection requires the use of a resource augmentation of

$\beta' = \frac{\sqrt{3}+1}{2}$ . However, in the Euclidean plane for instance, no resource augmentation is required to detect such cliques (whatever the underlying norm), i.e.  $\beta' = 1$ . Hence the following corollary (one can find its proof in [15]):

**Corollary 2.6:** Algorithm 2 is a  $(\frac{2}{5}, 3)$ -approximation algorithm for BCDC in the plane, whatever the underlying norm.

**D.**  $(\frac{1}{3}, 2)$ -approximation algorithm for BCDC in a tree-metric space

In this section, we present Algorithm 3, which is a version of Algorithm 1 dedicated to tree-metric spaces, such as those produced by the Internet embedding tool Sequoia [3], [4]. A tree-metric space is a metric space, where the points can be mapped as the leaves of a weighted tree and where the distance between two nodes corresponds to the length of the weighted path between those two nodes in the tree. In the context of tree-metric spaces, Algorithm 3 is a  $(\frac{1}{3}, 2)$ -approximation algorithm for BCDC.

To build bins, Algorithm 3 is based on a variant of Next-Fit-Decreasing that we call Next-Fit-Distance-Decreasing: given a node in the tree, this strategy consists in sorting the elements in the non-increasing order of their distance to this node. Then, elements are considered one by one in this order and placed in the current bin, until the weight of this bin overtakes 1. When a weight larger than 1 is reached, the bin is closed and a new one is opened.

Given a node  $x$  in the tree, we denote by  $\text{parent}(x)$  its parent, and by  $\text{children}(x)$  the set of its children. The root of  $T$  will be denoted by  $r$ .

To ease the description of Algorithm 3, we will use the following definition.

**Definition 2.4:** Given a tree  $T$  and a node  $e \in T$ ,  $\text{SUBTREE}_{d_{\max}}(e)$  is the subtree of  $T$  rooted in  $e$  such that  $\forall e' \in \text{SUBTREE}_{d_{\max}}(e), d(e, e') \leq d_{\max}$ .

---

**Algorithm 3**  $(\frac{1}{3}, 2)$ -approximation algorithm for BCDC in a tree-metric

---

- 1:  $U \leftarrow S$  // Elements that do not belong to any bin
- 2: BUILDINSUBTREE( $r$ )

Procedure BUILDINSUBTREE( $x$ ):

- 1: **for** each node  $y \in \text{children}(x)$  **do**
  - 2: BUILDINSUBTREE( $y$ )
  - 3: **end for**
  - 4: apply Next-Fit-Distance-Decreasing on  $U \cap \text{SUBTREE}_{d_{\max}}(x)$
  - 5: remove elements corresponding to each bin built from  $U$ .
- 

Implementing this algorithm in an efficient way just requires to perform an ascent of the tree. During this ascent, each node build a list of the nodes available in its subtree, at distance less than  $d_{\max}$  from itself, and orders this list in the

decreasing order of the distances from each node to itself. Then it can apply Next-Fit-Distance-Decreasing on such a list, to build bins. Eventually it sends the remaining elements of its list to its parent, after having got rid of the nodes too far from its parent to be used. This implementation is working  $O(n \log n)$ , and can also be efficiently distributed.

*Theorem 2.7:* Algorithm 3 is a  $(\frac{1}{3}, 2)$ -approximation algorithm for BCDC when  $d$  is a tree-metric space.

*Proof of Theorem 2.7:*

Given an instance  $\mathcal{I} = (S, d, w, d_{\max})$  of BCDC in which elements are embedded into a tree  $T$  having root  $r$ , we choose an optimal solution, denoted by  $\text{OPT}_{BCDC}$ . Let us denote by  $\mathcal{B}$  the set of bins built by Algorithm 3 on instance  $\mathcal{I}$ .

First, observe that each bin built has diameter at most  $2d_{\max}$ , hence the resource augmentation used. Indeed, two elements are put in a same bin if both are at distance less than  $d_{\max}$  from a given node and, thus, they are at distance at most  $2d_{\max}$  from each other.

Consider a bin  $B$  in a solution of BCDC. Let us denote by  $r(B)$  the smallest common ancestor of the nodes in  $B$ , called its root. Every element of  $B$  is at distance at most  $d_{\max}$  from  $r(B)$ .

Note that if two bins from two different solutions use the same element, then the root of one of the two bins is the ancestor of the root of the other bin.

We assign to each bin  $O$  of  $\text{OPT}_{BCDC}$  to the bin  $B$  of  $\mathcal{B}$  intersecting it, whose root is the furthest from the root of the tree  $T$ . If the maximal distance from the root of one of the bins of  $\mathcal{B}$  intersecting  $O$  to the root of tree  $T$  is the same for more than one bin, we choose the first bin built by Algorithm 3 among them. At least one bin  $B$  of  $\mathcal{B}$  intersects each bin  $O$ , since if no other bin has been built before, whereas at node  $r(O)$ , Algorithm 3 would have built a bin.

Let us first note that  $r(O)$  is an ancestor of  $r(B)$  (or  $r(B)$  itself) if  $O$  has been assigned to  $B$ . In fact, suppose it is not the case, then  $r(B)$  is a strict ancestor of  $r(O)$ , since they contain a common element. Algorithm 3 would have built a bin, when executed at node  $r(O)$ , since bin  $O$  itself could have been built before reaching node  $r(B)$  to build bin  $B$ . In fact, from the way  $O$  has been assigned to  $B$ , no element from  $O$  has yet been put into a bin of  $\mathcal{B}$  when Algorithm 3 reaches  $r(O)$  ( $r(B)$  is the "first" element to use elements from  $O$  to build a bin).

In what follows, we estimate the total weight available for  $\text{OPT}_{BCDC}$  to build bins. For this purpose, let us denote by  $w_{\text{lost}}$  the total weight of elements of  $\text{OPT}_{BCDC}$  not used in bins of  $\mathcal{B}$ . Each element whose weight is taken into account in  $w_{\text{lost}}$  belongs to an optimal bin. Thus, we can associate each of these elements to the bin of  $\mathcal{B}$  to which is assigned the optimal bin it belongs to. Then, given a bin  $B \in \mathcal{B}$ , we can define the lost weight associated to this bin,  $w_{\text{lost}}(B)$ , to be the total weight of elements belonging to

optimal bins assigned to  $B$  but not used in any bin of  $\mathcal{B}$ .

*Lemma 2.8:* For each bin  $B \in \mathcal{B}$ ,  $w_{\text{lost}}(B) < 1$ .

*Proof:* Let  $e_d$  be an element belonging to both  $B$  and a bin  $O$  of  $\text{OPT}_{BCDC}$  and such that  $r(O)$  is the closest possible from the root of  $T$ . Let  $d$  be the distance from  $r(B)$  to  $e_d$ . Let  $e_{d'}$  be an element of one of the optimal bins assigned to  $B$ , not belonging to  $B$ , not used in a bin of  $\mathcal{B}$ , and being the furthest from  $r(B)$ . Let  $d'$  be the distance from  $r(B)$  to  $e_{d'}$  (see Figure 2).

We have  $d \geq d'$ . In fact, when  $B$  is built at node  $r(B)$ , it uses elements by taking them in the non-increasing order of their distance to itself. Thus, it begins by using elements that are the furthest from itself. Since it uses  $e_d$  and not  $e_{d'}$ , then  $d(r(B), e_d) \geq d(r(B), e_{d'})$ .

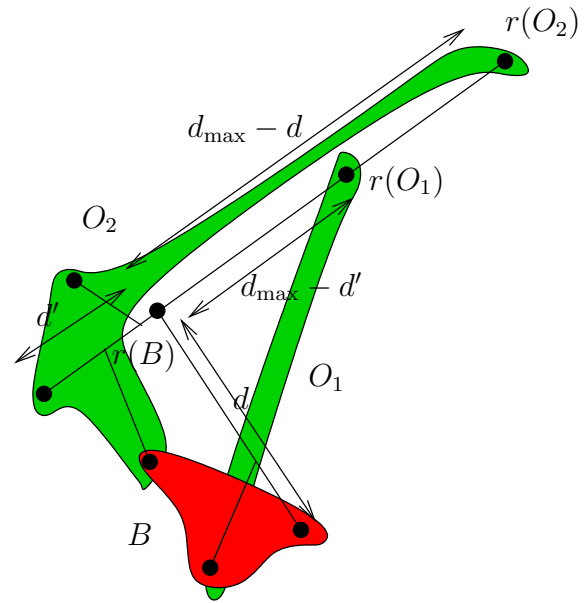


Figure 2. Example of a bin  $B$  of  $\mathcal{B}$  and two optimal bins  $O_1$  and  $O_2$  whose ancestors  $r(O_1)$  and  $r(O_2)$  respectively belong to bin  $O_1$  and  $O_2$ .  $r(O_2)$  is at distance at most  $d_{\max}$  from all the elements taken into account in  $w_{\text{lost}}(B)$ . Those elements are located between  $B$  and  $r(O_2)$ .

Each ancestor of an optimal bin assigned to  $B$  is an ancestor of  $r(B)$  (or  $r(B)$  itself). There exists at least one among them that is the ancestor of all the others. If it is the case for more than one, just arbitrarily choose one among them. This node,  $a$ , is at distance at most  $d_{\max} - d$  from  $r(B)$ , using the definition of  $d$ . Thus, it is at distance at most  $d_{\max} - d + d'$  from every element taken into account into  $w_{\text{lost}}(B)$ , from the definition of  $d'$ . Since  $d \geq d'$ , it is at distance at most  $d_{\max}$  from this set of elements.

Thus, all the elements taken into account in  $w_{\text{lost}}(B)$  are at distance at most  $d_{\max}$  from  $a$ . Thus  $w_{\text{lost}}(B) < 1$ , otherwise Algorithm 3 would have built a bin when at node  $a$  or before. ■

The weight of each bin of  $\mathcal{B}$  is strictly smaller than 2. We

can upper bound the cardinality of an optimal solution by

$$|\text{OPT}_{BCDC}| \leq w(\text{OPT}_{BCDC}) \leq 2|\mathcal{B}| + \sum_{B \in \mathcal{B}} w_{\text{lost}}(B) \leq 3|\mathcal{B}|,$$

hence the claimed approximation ratio. ■

### III. EXPERIMENTAL EVALUATION

In this section, we propose a comparison of the most widely encountered embedding tools in the literature, namely Vivaldi [1], [2] and Sequoia [3], [4].

As presented in Section II-C, Vivaldi associates to each node of a network two coordinates in the Euclidean plane plus a height. In such a space, the distance between two nodes having coordinates  $(a_x, a_y, a_h) \in \mathbb{R}^2 \times \mathbb{R}^+$  and  $(b_x, b_y, b_h) \in \mathbb{R}^2 \times \mathbb{R}^+$  is :  $d(a, b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2} + a_h + b_h$ . This embedding allows good prediction of the latencies between each pair of nodes [1], [2].

Sequoia embeds Internet nodes into one or several weighted trees in which each node is either a leaf or the root, and in which internal nodes are virtual nodes. The distance between two points in the original network is estimated to be well approximated by the distance in the embedding tree (or as the median of these distances in the case of multiple trees). In [3], it is claimed that the accuracy of the prediction is higher using Sequoia as soon as a few trees are used.

The main originality of the approach we propose is that it is done in the context of a specific application, heterogeneous resources clustering. Indeed, the accuracy of embedding tools have already been compared in [3], but the goal was only to estimate the closeness between latency predictions returned by the embeddings and the actual measurements.

In practice, when considering an actual application, the chain is more complicated and the performance of the whole chain has to be evaluated. Indeed, as we prove in this paper in the context of resource clustering (BCDC), the simplicity of the metric space where original data are embedded plays an important role in the design of the algorithms, and leads to different approximation ratios and inapproximability results. Therefore, to compare the respective performance of embedding tools, we will compare the results obtained using

- no embedding (direct use of the latency matrix) and Algorithm 1
- Vivaldi embedding and Algorithm 2
- 1-tree Sequoia embedding and Algorithm 3.

In this context, it is worth noting that the obtained classification strongly depends on the target application, since Vivaldi or Sequoia spaces may be very well suited to a particular application and not to another.

#### A. Experimental Protocol

In order to perform realistic simulations, we need to estimate the distance (the latencies) between any pair of re-

sources and we need to estimate the heterogeneous capacity of the resources.

In order to estimate latencies, we ran simulations on a real dataset taken from the Meridian project<sup>2</sup> containing all-pairs latency measurements between 2500 nodes arbitrarily chosen from PlanetLab<sup>3</sup>. In this matrix, latency measures are symmetric.

This matrix contains a lot of triangular inequality violations. The study in [5] makes a difference between triangular inequality violations due to the structure of Internet itself or due to the traffic on the Internet, and violations due to measurements' inaccuracy. Considering the study by Lebar *et al.* in [6], for each triple we consider that if the value  $\rho$  associated to this triple is larger than 10, then it might be due to the inaccuracy of the measurement.

Thus, in a first time, we compute for each latency measure between two points the number of times it appears in a triangle in which the  $\rho$  value is larger than 10. Figure 3 depicts the repartition function of this number: a point  $(x, y)$  means that  $y\%$  of the latency values appear in less than  $x$  triangles having a  $\rho$  value larger than 10.

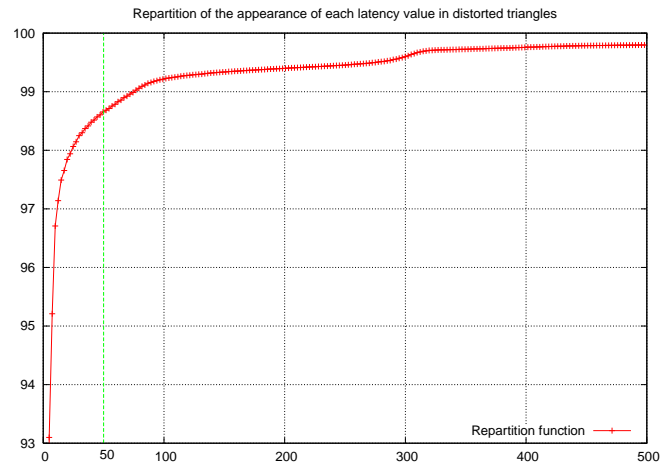


Figure 3. Repartition function of the number of triangles having a value of  $\rho$  larger than 10 in which appear each latency value.

Using Figure 3, we choose to modify a latency value as soon as it appears in more than 50 triangles in which the value of  $\rho$  is larger than 10, since only around 1% of latency values appear in more than 50 such triangles.

To modify each of these latency values, we examine all triangles it appears in, whatever its  $\rho$  value, compute in each triangle the sum of the two other sides, and affect to the latency value to be modified the minimum of these sum values. To obtain this "fixed matrix", we modified around 1% of the latency values in the original latency matrix. In

<sup>2</sup><http://www.cs.cornell.edu/People/egs/meridian/data.php>

<sup>3</sup><http://www.planet-lab.org/>

what follows, we run simulations using both the original latency matrix, and this “fixed matrix”.

In order to estimate the heterogeneous capacities of the resources, we rely on the XtremLab project <sup>4</sup> dataset. XtremLab consists in a list of the characteristics of nodes running BOINC [10] applications. We extract from it the data corresponding to the computing power made available by each node to the platform (their value in FLOPS). We get rid of values smaller than 500 KFlops and larger than 10 GFlops (getting rid of less than 0.1% of the values). We choose uniformly at random (so that the resulting distribution is the same as the original one) 2500 values among this set, and work with a total weight of 3632GFlops.

For each simulation, we rely on the following protocol

- We embed the latency matrix, using either Vivaldi or Sequoia algorithm (using a unique tree in the case of Sequoia).
- We apply to this embedding the corresponding algorithm for BCDC, all with the same set of weighted elements and for different values of  $d_{\max}$ , from the minimal distance previously identified to a value of 200ms (since beyond this value, no evolution is observed).

Algorithm 2 and Algorithm 3 dedicated to specific metric spaces (respectively Vivaldi and Sequoia-based) exhibit better theoretical approximation ratios than Algorithm 1, designed for the general case. Nevertheless, we keep Algorithm 1 for the sake of comparison, since a benefit of a better approximation ratio may be canceled by the lack of accuracy of the embedding tool. On the other hand, as mentioned in the introduction, it is unrealistic in practice to assume that the whole latency matrix is known. Indeed, in the context of large scale dynamic platforms, the time taken to estimate all latencies is too high with respect to the dynamics of the system. It is worth noting that the implementation of Vivaldi does not require a centralized knowledge of the matrix, which is not the case for Sequoia to the best of our knowledge. The last simulation we run is also based on Vivaldi. From this embedding, we re-compute the latency matrix (where the distance between any pair of nodes is their distance in Vivaldi space) and we run Algorithm 1, designed for general semi-metric spaces, but which offers better theoretical guarantee than Algorithm 2 dedicated to Vivaldi (since it uses a resource augmentation of 4 instead of  $2 + \frac{3\sqrt{3}+1}{2} \simeq 4.6$ ).

For each built cluster, we estimate its diameter as the maximal distance between any two nodes in the cluster, where distances are actual ones, i.e. those of the latency matrix, whatever the intermediate distance estimated by the embedding tool is. For each simulation and for each value of  $d_{\max}$ , we plot the average diameter of built bins, and the number of “valid” bins, i.e. bins whose diameter in the

latency matrix is lower than  $\beta d_{\max}$ , the maximum diameter allowed by the corresponding algorithm.

We simulate the execution of the different algorithms presented in the previous sections, dedicated to different metric spaces

- Algorithm 1 ( $(\frac{2}{5}, 4)$ -approximation algorithm presented in Section II-B, directly using a latency matrix to obtain distances),
- Algorithm 2 ( $(\frac{2}{5}, 2 + \frac{3\sqrt{3}+1}{2})$ -approximation algorithm presented in Section II-C, dedicated to a Vivaldi space),
- Algorithm 3 ( $(\frac{1}{3}, 2)$ -approximation algorithm presented in Section II-D),
- Algorithm 1 using a latency matrix based on an embedding using Vivaldi.

We ran simulations for different values of  $W$ , and choose  $W \simeq 11GFlops$  as the more meaningful and representative value (thus, each bin will require about 0.3% of the total weight). For smaller values of  $W$ , solutions tend to put fewer and fewer elements in each bin, since element weights get closer to the threshold. Thus, the number of built bins tends towards a steady value for a smaller value of  $d_{\max}$ . For larger values of  $W$ , the weight constraint becomes the most important and the problem is close to classical Bin Covering, since the distance constraint is less and less sensitive: even with large values of  $d_{\max}$ , bins can still be hard to build due to the lack of weight. Using such a weight threshold, bins contain on average approximately 7 elements.

## B. Simulation results

Figures 4 and 5 respectively depict, using each of the embedding/dedicated algorithm combinations, the number of valid built bins and the mean diameter of the built bins, for several values of  $d_{\max}$ , using the *original* latency matrix. Figures 6 and 7 respectively depict, using each of the embedding/dedicated algorithm combinations, the number of valid built bins and the mean diameter of the built bins, for several values of  $d_{\max}$ , using the *fixed* latency matrix.

In all cases, Algorithm 1 run on the original matrix performs better than both Algorithm 2 and Algorithm 1 using the latency matrix built from Vivaldi embedding. All of these outperform Algorithm 3.

Algorithm 2 and Algorithm 1 using the latency matrix built from Vivaldi embedding perform very similarly. It is worth noting that although Algorithm 2 requires a resource augmentation of  $\beta \simeq 4.6$ , the mean diameter of the bins it builds tends to be around the same values as the mean diameter of Algorithm 1, that only requires a resource augmentation of  $\beta = 4$ .

Vivaldi performs almost as efficiently as the general case, despite the loss of accuracy in the distance prediction induced by the embedding. The loss of accuracy due to the use of an embedding tool is therefore compensated by the design of an efficient algorithm in a simpler space. Moreover, this proves that using an embedding tool and a

<sup>4</sup><http://xw01.lri.fr:4320/>

dedicated algorithm provides as good results as when using a complete latency matrix, whereas obtaining such a latency matrix is impossible in practice.

By contrast we observe only minor differences in the results of Algorithm 1 and Algorithm 2 between the original and the “fixed” matrix. This suggests that these algorithms are much more robust.

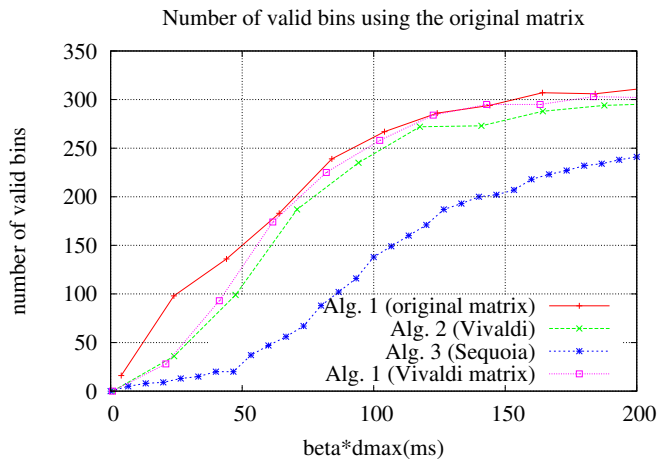


Figure 4. Number of bins built for BCDC, for different values of  $d_{max}$ , using different embeddings, using the original latency matrix.

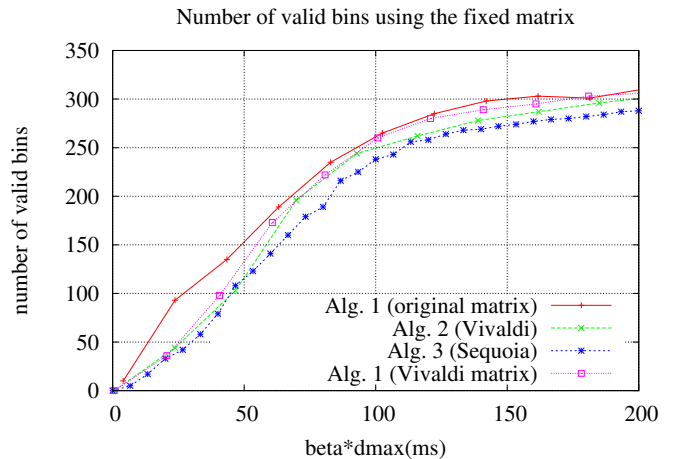


Figure 6. Number of bins built for BCDC, for different values of  $d_{max}$ , using different embeddings, using the fixed latency matrix.

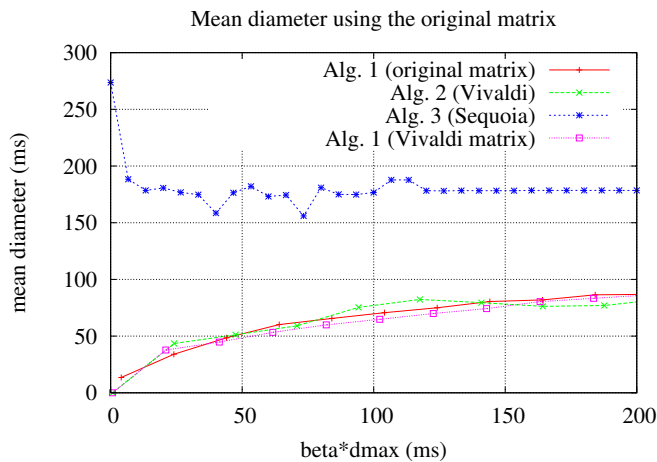


Figure 5. Mean diameter of bins built for BCDC, for different values of  $d_{max}$ , using different embeddings, using the original latency matrix.

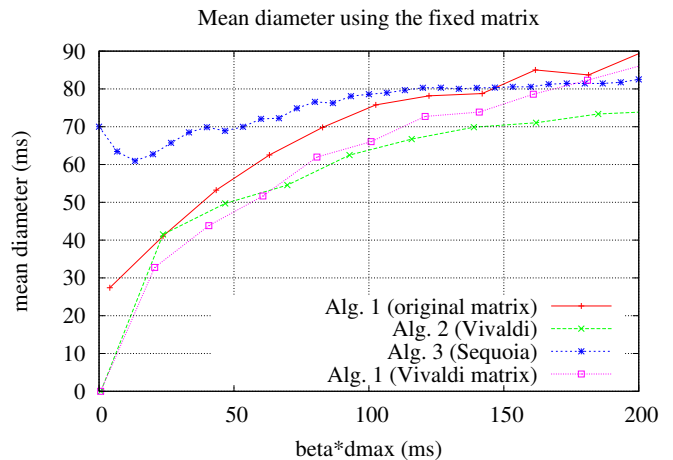


Figure 7. Mean diameter of bins built for BCDC, for different values of  $d_{max}$ , using different embeddings, using the fixed latency matrix.

Algorithm 1 and Algorithm 2 work in two phases. Figure 8 shows the percentage of bins built during the second phase for each simulation involving one of these algorithms, using the original matrix. This shows that in each case, the second phase is crucial to the algorithms’ performance.

Now we examine Figures 6 and 7. In these two Figures, Sequoia performs far better than on the original latency matrix, and the results of Algorithm 3 become similar to those of the other algorithms. This shows that our dedicated algorithm for BCDC on Sequoia (with one tree) is not robust, since a modification of only 1% of the latency values drastically changes its performances.

## REFERENCES

- [1] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris, “Practical, distributed network coordinates,” *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 1, pp. 113–118, 2004.
- [2] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, “Vivaldi: a decentralized network coordinate system,” *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 15–26, 2004.

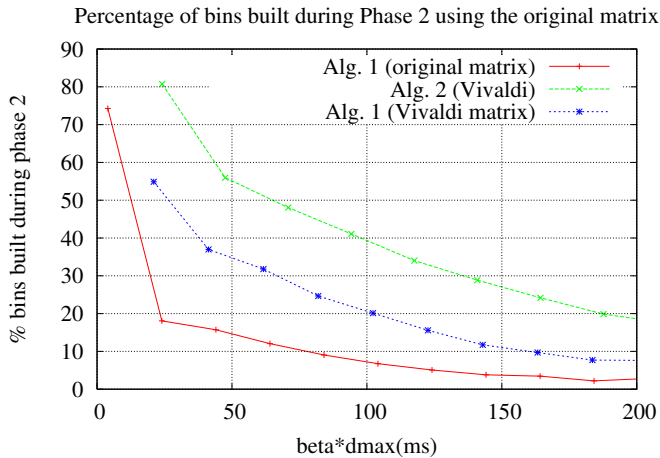


Figure 8. Percentage of bins built during Phase 2, for different values of  $d_{\max}$ , for each algorithm working in two phases, using the original latency matrix.

- [3] V. Ramasubramanian, D. Malkhi, F. Kuhn, M. Balakrishnan, A. Gupta, and A. Akella, "On the treeness of internet latency and bandwidth," in *SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM, 2009, pp. 61–72.
- [4] I. Abraham, M. Balakrishnan, F. Kuhn, D. Malkhi, V. Ramasubramanian, and K. Talwar, "Reconstructing approximate tree metrics," in *PODC '07: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM, 2007, pp. 43–52.
- [5] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee, "Triangle inequality variations in the internet," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, ser. IMC '09. New York, NY, USA: ACM, 2009, pp. 177–183. [Online]. Available: <http://doi.acm.org/10.1145/1644893.1644914>
- [6] E. Lebar, P. Fraigniaud, and L. Viennot, "The inframetric model for the internet," in *Proceedings of the 27th IEEE International Conference on Computer Communications (INFOCOM)*, Phoenix, 2008, pp. 1085–1093.
- [7] S. Assmann, D. Johnson, D. Kleitman, and J. Leung, "On a dual version of the one-dimensional bin packing problem," *Journal of algorithms*, vol. 5, no. 4, pp. 502–525, 1984.
- [8] O. Beaumont, N. Bonichon, P. Duchon, and H. Larchevêque, "Distributed approximation algorithm for resource clustering," in *SIROCCO 2008*, ser. Lecture Notes in Computer Science, A. A. Shvartsman and P. Felber, Eds., vol. 5058. Springer, 2008, pp. 61–73.
- [9] O. Beaumont, N. Bonichon, P. Duchon, and H. Larcheveque, "Distributed approximation algorithm for resource clustering," in *OPODIS 2008*, ser. Lecture Notes in Computer Science, vol. 5401. Springer, 2008, pp. 564–567.

- [10] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 4–10.
- [11] J. Csirik and G. J. Woeginger, "Resource augmentation for online bounded space bin packing," *J. Algorithms*, vol. 44, no. 2, pp. 308–320, 2002.
- [12] L. Epstein and R. van Stee, "Online bin packing with resource augmentation," *Discrete Optimization*, vol. 4, no. 3-4, pp. 322–333, 2007.
- [13] B. Clark, S. Colbourn David, and J. Charles, "Unit disk graphs," *Discrete mathematics*, vol. 86, no. 1-3, pp. 165–177, 1990.
- [14] L. Lovász and M. Plummer, *Matching theory*. Elsevier Science Ltd, 1986.
- [15] H. Larchevêque, "Agrégation de ressources avec contrainte de distance : applications aux plateformes de grande échelle(in french)." Ph.D. dissertation, Université de Bordeaux, 2010.

#### IV. BIOGRAPHIES

**Olivier Beaumont** received his PhD degree from the University of Rennes in 1999. Between 1999 and 2006, he was assistant professor at Ecole Normale Supérieure de Lyon and then at ENSEIRB in Bordeaux. In 2004, he defended his "habilitation à diriger les recherches" and was appointed as Senior Scientist at INRIA in 2007. His research interests focus on the design of parallel and distributed algorithms, overlay networks on large scale heterogeneous platforms and combinatorial optimization.

**Nicolas Bonichon** received his PhD degree from the Université Bordeaux 1 in 2002. He has been holding a position as assistant professor in Université Bordeaux 1 since 2004. His research interests include distributed algorithms, compact data structure, graph drawing and enumerative combinatorics.

**Philippe Duchon** received his PhD degree from Université Bordeaux 1 in 1998. Between 1999 and 2009, he was assistant professor at ENSEIRB in Bordeaux. Since 2009 he has been a professor of computer science in Université Bordeaux 1. His research interests range from enumerative combinatorics to random generation and the design and analysis of distributed and randomized algorithms.

**Hubert Larchevêque** received his PhD degree from Université Bordeaux 1 in 2010. His research interests include distributed algorithms, approximations algorithms, resource clustering and modelisation of Internet.