Massively Parallel Algorithms and Hardness for Single-Linkage Clustering under ℓ_p Distances (Appendicies)

Grigory Yaroslavtsev [1] **Adithya Vadapalli** [1]

A. Exact Hamming MST

Theorem A.1. *For $d = O(1)$ Hamming MST can be computed exactly in $O(\log n)$ rounds of MPC.*

Proof. For the special case $d = 2$ the algorithm is particularly simple and is given in Section A.0.1. For $d = O(1)$ we construct an auxiliary graph with integer edge weights in the interval $[1, \dots, d]$ and use MPC algorithm graph connectivity to compute MST for it.

For a binary vector $b \in \{0,1\}^d$ of Hamming weight t and $v \in \mathbb{R}^d$ we use notation $v(b)$ to denote a vector in $\mathbb{R}^t$ consisting of coordinates of v corresponding to non-zero values of b. For a binary vector $b \in \{0,1\}^d$ we define an order relation $\leq_b$ on vectors in $\mathbb{R}^d$ as follows: $v_1 \leq_b v_2$ if and only if $v_1(b) \leq v_2(b)$ and $\leq$ is the lexicographic order.

- Initialize G to an empty graph on n vertices and $\mathcal{F}$ to a forest of singleton vertices.
- For each binary vector $b \in \{0,1\}^d$:
  - Sort input vectors $v_1, \dots v_n$ according to $\leq_b$ breaking ties arbitrarily.
  - Let $\pi(i)$ be the index of the i-th vector in this sorted order.
  - For $i = 1, \dots, n-1$ create an edge in G of weight $d - \|b\|_0$ between vertices $\pi(i)$ and $\pi(i+1)$ if $v_{\pi(i)}(b) = v_{\pi(i+1)}(b)$
- For $i = 1, \dots, d$
  - Augment $\mathcal{F}$ using edges of length i in G to a spanning forest for the subgraph of G consisting of edges of weight at most i.

The first loop of the above reduction can be performed in $O(1)$ rounds of MPC by replicating the data $2^d = O(1)$ times and running $O(1)$-round MPC sorting algorithm (Goodrich et al., 2011) on all the replicas in parallel. The second loop can be performed in $O(d \log n)$ rounds total by using $O(\log n)$ connectivity algorithm in each iteration. Correctness of the above algorithm follows from the following observation: if $\|v_i - v_j\|_0 = t$ then there exists a path in the graph G between i and j that uses only edges of weight at most t.

Indeed, if Hamming distance between two vectors equals t then there exists a subset of $d - t$ coordinates where these two vectors agree. Let $b \in \{0,1\}^d$ be the indicator vector of this subset. In the iteration of the first loop corresponding to b let p_i and p_j be positions of vectors v_i and v_j in the sorted order in this iteration. W.l.o.g $p_i < p_j$ and for all $k = p_i, \dots, p_j - 1$ we added an edge of weight t between $\pi(k)$ and $\pi(k+1)$ creating the desired path in G.

From the above observation it follows directly that for all $t = 1, \dots, d$ the number of connected components in the subgraph of G induced by edges of weight at most t is the same as the number of connected components induced by edges of Hamming weight at most t in the original input. Thus executions of Kruskal's algorithm on G and the distance graph under Hamming distance give the same result and hence the MST constructed by the algorithm above is optimal.

A.0.1. Simple proof of Theorem A.1 for $d = 2$

An instance of Hamming MST for $d = 2$ is represented by n vectors $(x_1, y_1), \dots, (x_n, y_n)$. Because all edges in the distance graph have cost either 1 or 2 the cost of the optimum MST equals to $n + c - 2$ where c is the number of connected components in the subgraph induced by edges of cost 1. We will construct a subgraph that has the same set of connected components using only $2n$ edges and then run an $O(\log n)$-round MPC connectivity algorithm on it.

Formally the construction is given as follows. First, we create a vertex i for each input vector (x_i, y_i). Then we create edges between these vertices as described below, repeating this process with the role of x and y coordinates flipped

*Equal contribution [1]Department of Computer Science, Indiana University, Bloomington, Indiana, United States. Correspondence to: Grigory Yaroslavstev <grigory@grigory.us>, Adithya Vadapalli <avadapal@iu.edu>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

(i.e. sort and group according to y).

- Sort input vectors according to the x-coordinate and then according to the y-coordinate.
- For each value of x let $y_1^x, \dots, y_t^x$ be the corresponding sorted y-coordinate values.
- For all j where $1 \le j < t$ create an edge between vertices representing (x, y_j^x) and (x, y_{j+1}^x) in the input graph.

□

This reduction can be performed in a constant number of rounds of MPC using a constant-round MPC sorting algorithm of (Goodrich et al., 2011). Furthermore, it preserves the connected components in the graph induced by edges of length 1 under Hamming distance. Indeed, such edges correspond to pair of vectors that have one of the coordinates being equal and hence in our construction there is a path between vertices representing such vectors. This completes the proof for the case $d = 2$.

B. Implementation details and performance analysis

The implementation details in this section are similar to (Andoni et al., 2014). The main difference is that we need the Solve-and-Sketch to work under ℓ_1 and ℓ_∞ rather than just ℓ_2 as in (Andoni et al., 2014) which requires some modifications in the analysis.

B.1. Distance-preserving partitions

We use the following construction of (Andoni et al., 2014) to build a distance-preserving partition $\mathcal{P}$ for $S \subseteq \mathbb{R}^d$. We can always shift S such that all points fit into a box $[0, \Delta]^d$ where Δ is the diameter of the metric space(S, ℓ_∞^d). Pick a vector $r \in \mathbb{R}^d$ uniformly at random from $[0, \Delta]^d$. Two points u and v belong to the same cell at level $\ell \in \{0, \dots, L\}$ if and only if for all dimensions $i \in [d]$ it holds that $\lfloor \frac{(u_i - r_i)\alpha^{L-\ell}}{\Delta} \rfloor = \lfloor \frac{(v_i - r_i)\alpha^{L-\ell}}{\Delta} \rfloor$ where α is a parameter (see Figure 1 for an example). Note that this partition is indexable since coordinates of the point $x \in \mathbb{R}^d$, random shift r and ℓ suffice for computing $C_\ell(x)$.

Lemma B.1 ((Andoni et al., 2014), Lemma 5.3)**.** *Indexable randomized hierarchical partition $\mathcal{P}$ given by the construction above has $L = O(\log_\alpha |S|)$ levels can be constructed in $O(1)$ rounds of MPC and is an $(1/\alpha, d, (\alpha + 1)^d)$-distance-preserving partition for (S, ℓ_2^d) with approximation $\gamma \le \sqrt{d}$.*

Below we show that this partition $\mathcal{P}$ is also distance-preserving for ℓ_1^d and ℓ_∞^d.

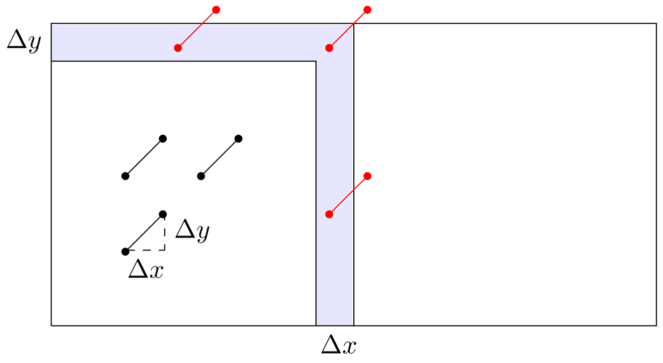

Figure 1. Probability of a cut proportional to $\Delta x + \Delta y$ corresponds to edge landing in the shaded region

Lemma B.2. *Indexable randomized hierarchical partition $\mathcal{P}$ is: 1) $(1/\alpha, d^2, (\alpha+1)^d)$-distance-preserving for ℓ_1^d with approximation d, 2) $(1/\alpha, d, (\alpha+1)^d)$-distance-preserving for ℓ_∞^d with approximation 1.*

Proof. The degree bound of $(\alpha + 1)^d$ follows from Lemma B.1 in both cases so in the rest of the proof we only analyze other parameters of the partition.

Part 1. Under ℓ_1 we have $\Delta(S) \ge \Delta$ and $\Delta(S) \le d\Delta$. By construction cells of the partition at level ℓ have diameter at most $d\Delta\alpha^{\ell-L}$. Hence we can set $\gamma = d$ and $\Delta_\ell = d\alpha^{\ell-L}\Delta(S)$ which satisfies the bounded diameter condition $\Delta(P_\ell) \le \Delta_\ell$. To verify the condition on probability of cutting an edge consider two points $u, v \in \mathbb{R}^d$. The probability that $\lfloor \frac{(u_i - r_i)\alpha^{L-\ell}}{\Delta} \rfloor \ne \lfloor \frac{(v_i - r_i)\alpha^{L-\ell}}{\Delta} \rfloor$ for a fixed i is at most $\frac{\|u_i - v_i\|_1 \alpha^{L-\ell}}{\Delta}$. By a union bound the probability that u and v belong to different cells at level ℓ is at most

$$\frac{\|u - v\|_1 \alpha^{L-\ell}}{\Delta} = \frac{d\|u - v\|_1 \Delta(S)}{\Delta_\ell \Delta} \le \frac{d^2\|u - v\|_1}{\Delta_\ell}.$$ □

Part 2. Under ℓ_∞^d we have $\Delta(S) = \Delta$. By construction cells of the partition at level ℓ have diameter at most $\Delta\alpha^{\ell-L}$. Hence we can set $\gamma = 1$ and $\Delta_\ell = \alpha^{\ell-L}\Delta$. As in the previous case the probability that two vectors $u, v \in \mathbb{R}^d$ belong to different cells of the partition is at most $\frac{\|u-v\|_1 \alpha^{L-\ell}}{\Delta}$ which can be bounded as follows:

$$\frac{\|u - v\|_1 \alpha^{L-\ell}}{\Delta} \le \frac{d\|u - v\|_\infty \alpha^{L-\ell}}{\Delta} = \frac{d\|u - v\|_\infty}{\Delta_\ell}.$$

Thus $\mathcal{P}$ is indeed an $(1/\alpha, d, (\alpha+1)^d)$-distance-preserving partition for ℓ_∞^d with approximation 1.

C. Solve-and-sketch details

Below we first show in Lemma C.1 that if the unit step can be executed efficiently then the overall computation also can. This theorem is analogous to Theorem 5.2 in (Andoni et al., 2014) but here we give a simpler and faster implementation using the fact that all our partitions are indexable. Then we proceed to describe implementations of Algorithm 3 for ℓ_1 and ℓ_∞ in Lemma C.3. For ℓ_2 such an implementation is given in (Andoni et al., 2014), Lemma 3.23 and for ℓ_1 and ℓ_∞ the implementation is analogous using approximate nearest neighbor search (Arya et al., 1998) and appropriate $\epsilon^2\Delta_\ell$-covering construction for each metric.

Lemma C.1. *Let $t_u(x)$ be a convex function and $s_u(x)$ be a function with at least linear growth, i.e. $s_u(x) \geq x$. Let $P = (P_0, \dots, P_L)$ be an indexable partition labeling $M(S, \rho)$ with L levels sampled from a randomized (a, b, c)-distance-preserving family $\mathcal{P}$. Let $\mathcal{A}_u$ be a unit step algorithm which for an input of size n_u takes time at most $t_u(n_u)$, uses space $s_u = s_u(n_u)$ and produces output of size $p_u = O(\min(p, n_u))$ where p is a parameter. If space per machine is s and $s_u(cp_u) \leq s/3$ then the unit step computations for all cells of the partition can be executed in $O(L)$ rounds of MPC on $O(n/s)$ machines. Furthermore, local computation time per machine in every round is bounded by $t_u(s)$.*

Proof. We process the partition level by level assuming that when we process level k all results for level $k-1$ are already computed. Furthermore, all results from the previous round are labeled by cells of the partition at P_k that they belong to. Thus in round k we just execute the unit step for all cells in this level and label the results with the cell they belong at level $k+1$. The latter part can be performed locally using the fact that P is an indexable partition. Note that since $p_u = O(\min(p, n_u))$ the overall output produced in each round has size $O(n)$.

Proposition C.2. *Total number of machines that we need to execute each round is at most $O(n/s)$ and maximum time per machine in a round is at most $t_u(s)$.*

Proof. First we estimate total space we need to allocate on machines. Let m_i be the number of nonempty subcells of the i-th cell in the k-th level, i.e. the number of inputs for this cell. For each i and $j = 1, \dots, m_i$ let n_{ij} be the size of output produced by the j-th subcell in the previous round that is now input for the i-th cell in k-th level.

We know that $\sum_{i,j} n_{ij} = O(n)$ and $n_{ij} \leq p$. The input to $\mathcal{A}_u$ has overall size at most cp and hence each execution uses space at most $s_u(cp) \leq s/3$. Note that using the fact that $s_u(x) \geq x$ this also implies that the input for each job is of size at most $s/3$.

We assign executions of $\mathcal{A}_u$ jobs in round k arbitrarily to machines in such a way that we only start a new machine if there is no existing machine with at least $2s/3$ space available. This ensures that in the end of the assignment process each machine has at least $s/3$ unused space for executing the jobs. We then execute jobs assigned to each machine sequentially using this space and store all inputs for all jobs locally. Note that our assignment process ensures that if $\mathcal{S}$ is the total space required to store inputs for all jobs then the total space we use is at most $3\mathcal{S} + s = O(n)$ and the total number of machines is at most $3\mathcal{S}/s + 1 = O(n/s)$.

Suppose we are using t machines in total in this round and let B_i be the subset of jobs assigned to i-th machine. The maximum time per machine required to execute jobs in k-th round is then:

$$\begin{aligned}\max_{i=1}^{t}\left[\sum_{j\in B_i} t_u\left(\sum_{\ell=1}^{m_j} n_{j,\ell}\right)\right] &\leq \max_{i=1}^{t}\left[t_u\left(\sum_{j\in B_i}\sum_{\ell=1}^{m_j} n_{j,\ell}\right)\right]\\ &\leq t_u(2s/3)\\ &\leq t_u(s)\end{aligned}$$

□

Lemma C.3. *If the input metric space M is a subset of ℓ_1, ℓ_2 or ℓ_∞, the unit step Algorithm 3 has space complexity $s_u(n_u) = n_u \log^{O(1)} n_u$ words, time complexity $t_u(n_u) = (d/\epsilon)^{d+1} n_u \log^{O(1)} n_u$ and output size $p_u = O(\min((1/\epsilon)^{2d}, n_u))$ words.*

Proof. For ℓ_2 the proof is given in Lemma 3.23 of (Andoni et al., 2014). For ℓ_1 and ℓ_∞ the proof is analogous and follows from the fact that we can execute Algorithm 3 by using approximate nearest neighbor search. Details are analogous to the details for ℓ_2 and are given in (Indyk, 2000; Andoni et al., 2014). In particular Theorem 3.27 in (Andoni et al., 2014) describes how to use approximate nearest neighbor data structure of (Arya et al., 1998) for ℓ_2. Since (Arya et al., 1998) gives data structures with the same performance for ℓ_1 and ℓ_∞ as well the analysis of time and space performance in these two cases is the same.

The bound on the output size follows from the fact that we can construct an $\epsilon^2\Delta_\ell$-covering by imposing a grid with step size ξ and taking one arbitrary point from each cell of the grid (if it is nonempty). Fix $\xi = \epsilon^2\Delta_\ell/d$ for ℓ_1 and note that the total number of cells in the grid is at most $(\Delta_\ell/d\xi)^d$. Fix $\xi = \epsilon^2\Delta_\ell$ for ℓ_∞ and note that the total number of cells in the grid is $(\Delta_\ell/\xi)^d$. In both cases the bound of $(1/\epsilon)^{2d}$ on the size of the output follows. □

Putting things together we get the proof of Theorem 2.1:

Proof of Theorem 2.1. The algorithm is given as Algorithm 2.1 and hence the approximation guarantee follows from Theorem 2.5. Hence it only remains to analyze performance of Algorithm 2.1. Recall that this algorithm uses an (a, b, c)-distance-preserving partition where $a = 1/(s^{\alpha/d} - 1)$, $b = poly(d)$ and $c = s^{\alpha}$. Also recall that we set $\epsilon = \min\left(\frac{\eta}{6c_1 Lb}, \frac{\eta}{3c_2}\right)$ and hence if we set $\kappa = \min\left(\frac{1}{6c_1 Lb}, \frac{1}{3c_2}\right)$ then $\kappa\eta = \epsilon$ and κ is a constant because c_1, c_2, L and $b = poly(d)$ are constants.

A distance-preserving partition in Step 2 can be constructed for ℓ_2 using Lemma B.1 and for ℓ_1 and ℓ_∞ using Lemma B.2. This step takes $O(1)$ rounds of MPC under the resource constraints of the theorem. Thus, it suffices to show that Step 2 can be executed with required performance guarantees. Note that $p_u = O(\epsilon^{-2d}) = O((\kappa\eta)^{-2d}) \le s^{1-2\alpha}$ and hence $cp_u = O(s^{\alpha} s^{1-2\alpha}) = O(s^{1-\alpha})$. Using the assumption that ϵ, α and d are constants by Lemma C.3 we have $s_u(cp_u) = \tilde{O}(s^{1-\alpha}) \le s/3$ for sufficiently large s. Thus combining Lemma C.3 and Lemma C.1 and using the assumption that ϵ and d are constant we conclude that Step 2 can be executed on $O(n/s)$ machines with local computation time per machine in each round bounded by $\tilde{O}(s)$. Furthermore since the distance-preserving partition used in out algorithm has $O(\log_{s^{\alpha/d}} n) = O(d/\alpha \log_s n) = O(1)$ levels (by Lemma B.1) from Lemma C.1 it follows that this step only takes $O(1)$ rounds of MPC. Overall, each iteration of the loop in Algorithm 2.1 can be executed in $O(1)$ rounds of MPC and hence sequential execution takes $O(\log n)$ rounds as desired.

Finally, Boruvka's algorithm in Step 2 is run on $O(n \log n)$ edges and can be implemented in $O(\log n)$ rounds using $\tilde{O}(n/s)$ machines under constraints of the theorem. □

□

D. Experimental results

Figure 2 shows dependence of approximation to the k-SLC objective as a function of k for $\eta = 0.5$. Figure D shows how approximation varies empirically as a function of η (for $k = 10$). Evaluation of time performance demonstrates more than an order of magnitude speedup over sequential Prim's algorithm[1] for $k = 10$ as a function of η parameter and empirical approximation. Results are given for two different cluster setups: 1m/7w (Figure 4 and Figure 5) and 1m/3w (Figure 6 and Figure 7) and are averaged over multiple runs to ensure consistency. We note that dramatic increase in speedup for the KDDCUP04 dataset around value $\eta = 0.5$ and approximation 1.15 corresponds to the fact that local inputs start to fit in L2-cache which provides more than an order of magnitude improvement over RAM.

[1]We use Prim's algorithm as a sequential benchmark as our local computation steps are also performed using Prim's algorithm in the experiments. For datasets of our size Prim's algorithm outperformed approximate nearest-neighbor (ANN) based algorithms locally and it was our goal to use the best local algorithm. In order to observe performance improvement from using ANN algorithms larger datasets are required due to several log-factors and a large constant in the theoretical almost-linear time complexity.

E. Hardness

Part 2. We perform reduction as in Part 1 but without Step 3 and setting $\xi = 1$. Note that since the resulting vectors have at most 3 non-zero entries each the input can be represented in $O(n)$ space. A calculation similar to the above shows that in this case if there is an edge (i, j) in the graph then $\|v'_i - v'_j\|_1 = 2|1 - \xi| + 2|\xi|$. Otherwise, $\|v'_i - v'_j\|_1 = 2 + 4|\xi|$. The ratio between these two cases is maximized when $\xi = 1$ and equals 3.

Part 3. Given an instance $G(V, E)$ of sparse connectivity we reduce it to ℓ_2-2-SLC as follows. Let $n = |V|$ and $m = |E|$. We can assume that G has no isolated vertices as connectivity instances containing such vertices can be solved in $O(1)$ rounds of MPC by identifying isolated vertices.

For the i-th edge of the input we create a vector $v'_i \in \mathbb{R}^n$. We set $v'_{i,j} = 1$ if the i-th edge in the input is adjacent on vertex j and $v'_{ij} = 0$ otherwise. Then we apply JL-transform to reduce the dimension of constructed vectors to $O(\log n/\epsilon^2)$ obtaining vectors $v_1, \ldots, v_m$ as in Part 1.

Since G has no isolated vertices it is connected if and only if the set of its edges forms a connected subgraph. By construction if two edges i and j share a vertex then $\|v'_i - v'_j\| = \sqrt{2}$, otherwise $\|v'_i - v'_j\| = 2$. Hence, if G is connected all edges in the ℓ_2-MST for $v_1, \ldots, v_m$ have length $\sqrt{2}$. Otherwise, there exists an edge in ℓ_2-MST of length 2. Hardness of $(\sqrt{2} - \epsilon)$-approximation hence follows from the fact that JL-transform preserves all distances up to $(1 \pm \epsilon)$-approximation.

Part 4. We use the same reduction as in Part 3 but without using the JL-transform in the end. Since the instance of sparse connectivity has $O(n)$ edge we obtain $O(n)$ vectors with 2 non-zero entries in each. Hence, resulting instance can be stored in $O(n)$ space. As in Part 3 note that if two edges i and j share a vertex then $\|v_i - v_j\|_1 = 2$, otherwise $\|v_i - v_j\| = 4$. Hence the costs of ℓ_1-2-SLC differ by a factor of 2 depending on whether G is connected or not.

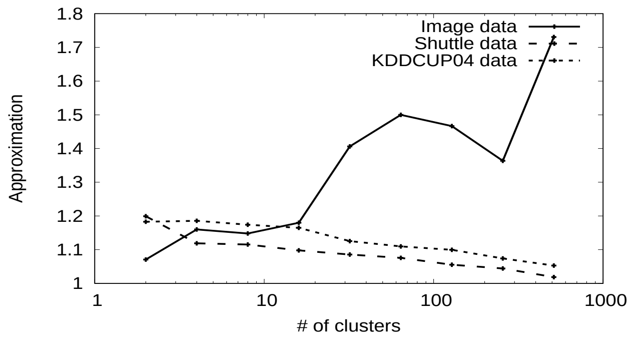

Figure 2. Approximation vs number of clusters

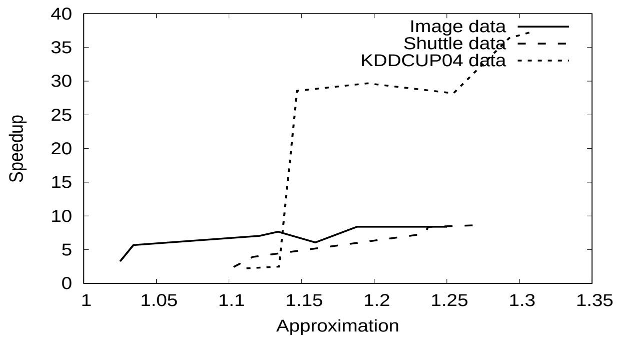

Figure 5. Speedup vs approximation, 1m/7w cluster

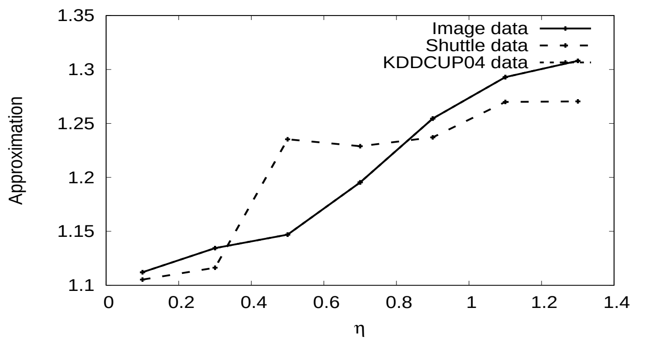

Figure 3. Approximation vs η parameter

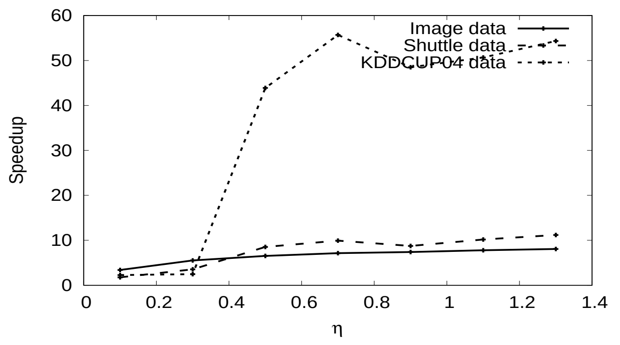

Figure 6. Speedup vs η parameter, 1m/3w cluster

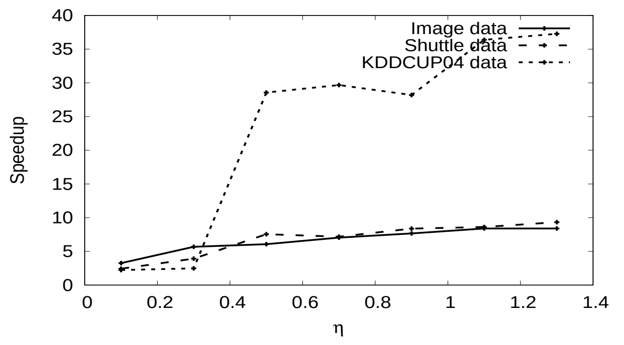

Figure 4. Speedup vs η parameter, 1m/7w cluster

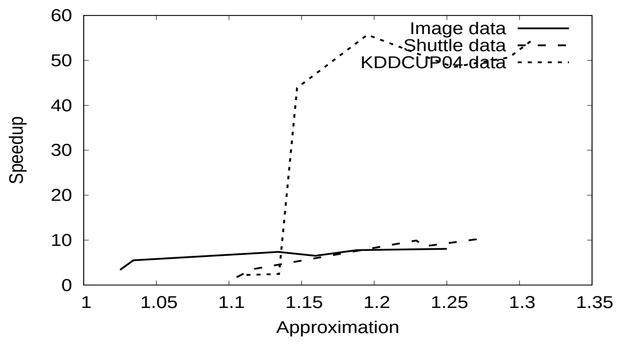

Figure 7. Speedup vs approximation, 1m/3w cluster

References

Andoni, Alexandr, Nikolov, Aleksandar, Onak, Krzysztof, and Yaroslavtsev, Grigory. Parallel algorithms for geometric graph problems. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pp. 574–583, 2014. URL http://arxiv.org/abs/1401.0042.

Arya, Sunil, Mount, David M., Netanyahu, Nathan S., Silverman, Ruth, and Wu, Angela Y. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

Goodrich, Michael T., Sitchinava, Nodari, and Zhang, Qin. Sorting, searching, and simulation in the mapreduce framework. In *Algorithms and Computation - 22nd International Symposium, ISAAC 2011, Yokohama, Japan, December 5-8, 2011. Proceedings*, pp. 374–383, 2011.

Indyk, Piotr. *High-dimensional Computational Geometry*. PhD thesis, Stanford University, 2000.