Cluster-Swap : A Distributed K-median Algorithm for Sensor Networks

Yoonheui Kim Victor Lesser Deepak Ganesan Ramesh Sitaraman
Computer Science Dept.
University of Massachusetts,Amherst
Amherst, MA 01002

Abstract

In building practical sensor networks, it is often beneficial to use only a subset of sensors to take measurements because of computational, communication, and power limitations. Thus, selecting a subset of nodes to perform measurements whose results will closely mirror the results of having all the nodes perform measurements is an important problem. This node selection problem, depending on the character of the function that integrates measurements and the type of measurements, can be mapped into a more general problem called the k-median problem. In the k-median problem we select a centroid set - a subset of nodes - that minimizes the function, that is the sum of the minimal costs between each node and a node in the centroid set. The set of selected nodes is called "centroids" or "leader nodes", where the cluster of a leader node is defined by the set of nodes closest to the leader node. We develop an approximate k-median distributed algorithm called Cluster-Swap, which does not require significant computational power, and does not require every node to know its exact position in the n-dimensional space but only its relative location in relation to a subset of nodes. In addition, Cluster-Swap limits communication costs and is flexible to network changes. The locally optimal solution reached by our algorithm is an approximation whose error is bounded by the maximum cost and number of nodes in the cluster. The error bound gives a tighter bound than other similar algorithms, given that the random initial solution is within a described reasonable range. We empirically show that the solution given by our distributed algorithm is close to both the approximate solution generated by the cited Local search heuristics and also the globally optimal solution while using fewer resources.

1. Introduction

Many interesting problems in sensor networks and traditional routing networks can be reduced to classic algorithmic problems such as the Traveling Salesman Problem (TSP), k-median, or k-center problem [5, 13, 12]. For example, Meliou *et al.* solve a data collection problem by reducing it to a TSP [12]. Das *et al.* solve the problem of aggregating sensor measurements by reducing it to a k-median or k-center problem [5]. The gateway placement problem is solved by also reducing it to a k-median problem [13] .

In sensor networks, there are many problems that can be reduced to the k-median problem, such as the subset selection problem [6, 9, 14] and the sensor measurement aggregation problem [2, 5], where a subset of size k that optimizes a certain cost function should be obtained. In [5], the error of the measurements is bounded by the k-median problem's cost function when only k sensors take measurements, assuming the measurements are correlated with distance. The error in the average value of measurements is bounded by $\frac{1}{n}\Sigma_{i \notin S} c(i, S)$, where S is the selected subset of size k, $c(i, S)$ is the distance between sensor i and set S therefore reducing the problem of minimizing the worst case error to a k-median problem.

However, the k-median algorithms developed so far have been focused on solving problems with large data sets that need to be clustered. These algorithms do not accommodate domains where there is a much smaller set of data points, limited computation power, and a lack of information about the absolute locations of data points. For example, in these settings each node may only know the relative location of nearby nodes. Furthermore, even for relatively small sets, exhaustively computing the optimal answer for the k-medain problem is not practical and thus requires approximation. Additionally, in a sensor network, it is often not feasible to send necessary information to a central location to solve. We develop a fully distributed k-median algorithm called Cluster-Swap that satisfies the constraints of this particular environment. The algorithm does not require extensive computing power, reduces communication load, and adjusts easily the solution when a node fails.

The Cluster-Swap algorithm presented here is based on the Local search heuristics [1] that only uses swap operations, where a swap is defined as switching the role between a centroid and a non-centroid node. Local search heuristics can be simply adapted to a distributed environment since the swap operation is very simple and can be done in a local manner. However, swapping two nodes in a distributed environment can have high communication cost if they are far apart. Therefore, we have created Cluster-Swap where the swap operations are performed in a limited context involv-

978-0-7695-3801-3/09 $26.00 © 2009 IEEE
DOI 10.1109/WI-IAT.2009.178

ing only the subsets of nodes that are relatively close. Also, using a similar approach to bound the error as in [1], we bound the error with quantities calculated from the solution of the algorithm, and show that the algorithm gives a reasonable bound that is confirmed in the experimental section. In addition, in the experimental section, we compare our algorithm with the Local search heuristics and Neighbor-Swap algorithm, which only swaps with directly connected neighbors. We provide experimental results showing that the solution quality of Cluster-Swap is close to that of the Local search heuristics with a 5-approximate solution without requiring significantly more resources than the Neighbor-Swap algorithm.

The paper is organized as follows. We first provide a formal description of the k-median problem and some known algorithms for its solution. We then provide a detailed description of the Cluster-Swap algorithm, the proof of an error bound, and implementation details. Finally, we show experimental results comparing Cluster-Swap to Local search heuristics and Neighbor-Swap to show the validity and efficiency of our approach.

1.1. Problem Description

Our goal is to solve the k-median problem without knowing the full structure of a network and without sending all network information to a central location. We consider the general case where there is a graph G=(E,V) and the cost function $C : E \rightarrow \Re$ on each edge. Since each edge is Euclidean, the triangle inequality holds. Within this setting, we search for a set of centroid nodes $\bar{T} \subset V$ where the sum of the cost function between each node and closest $t \in \bar{T}$ is minimized:

$$\bar{T} = argmin_T(\sum_{v \in V} min_{t \in T} c(v,t)) \quad (1)$$

We assume that each agent has the knowledge of the edges shared with direct neighbor nodes and can directly communicate at least with them. Given this problem description, we now present our algorithm and its performance bound.

1.2. Related Work

K-median clustering is a common technique used in the machine learning community to analyze data. It is known that the problem is NP-hard, and thus many approximation algorithms have been developed. Traditionally, research has been focused on how to solve k-medians for a very large number of points. Recently, there has been a growing interest in reducing problems to k-median problems. However, the algorithms developed so far do not suit the growing needs of those domains.

There is recent research in selecting a subset of input points to compute the k-median [8]. Unfortunately, the number of points needs to be very large to benefit from this approach, which mitigates its use for most sensor networks. Other relevant work includes k-median clustering for large data streams where it is impractical to store the whole stream before initiating processing. The k-median algorithms on streams [3, 7] are relevant to our study in that they work in a sequential order on the input stream and do not use much space. However the algorithms only guarantee $O(1)$ or higher approximation whose constant factors are not as small as desired. Additionally, some well studied linear programming algorithms are also not suitable as they require extensive computation, which is generally assumed to not be available [4].

There are also distributed versions of the k-median algorithm focused on data sets that are too large to store in one place. The goal of these distributed algorithms is to achieve a result close to the case when the data is centralized, and the algorithms focus on selecting a subset of data points in each location in order to efficiently aggregate the distributed partial results. There also is a centralized algorithm only applied to the Euclidean plane that achieves a $1+\epsilon$-approximate solution in $O(nkn^{O(1/\epsilon)} \log n)$ time [11]. This is a known best optimal guarantee but the fact that it only holds on a 2-dimensional space, it requires exhaustive computation power, and it assumes that the exact location of each point is known, make this approach inappropriate for our problem domain.

One of the simple and good approximation algorithms is based on Local search heuristics [1], and is known to guarantee a 5-approximate solution. Since this approach is simple, has a reasonable bound, and is easily applicable to a distributed environment, our algorithm is built based on this work.

2. The Cluster-Swap algorithm

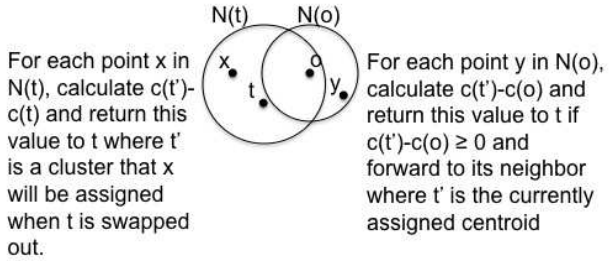


Figure 1. The range of swap(t, o) and cost change based on the location of nodes. The node t is swapped out and the node o is inserted where N(t) are the members of the cluster t before the swap and N(o) are the members of the cluster who are assigned to o after the swap, and c(x) is cost of a node x.

As described in the previous section, we seek to solve the k-median problem in a decentralized manner by swap operations as in the Local search heuristics, but limit the target to swap to only the non-centroids *in the same cluster*. The Cluster-Swap algorithm determines the *k* centroids that minimize the sum of the distances to the closest centroid where *k*, the number of centroids, is given as a parameter. This initial set of centroids is given as a parameter to reduce

the difficulty of getting the consensus in the network on the centroids and can be determined randomly.

We designed our algorithm based on the following assumptions. First, we assume that each node knows the cost function for each of its directly connected neighbors. This cost can be either the distance or another measure that satisfies the triangular inequality. Additionally, we assume that the initial set of centroids is pre-determined.

Given the assumptions, each node performs the same local algorithm on each cycle depending on whether it is a centroid or a non-centroid. At each cycle, the current k centroid nodes try to improve the solution quality by communicating with non-centroid nodes within the same cluster until no centroid can improve its cost through swaps, thus there is no change in centroid nodes.

The swap operation is taken in two steps, *test-swap(t,o)* and *swap(t,o)*. *test-swap(t,o)* tests the cost benefit of the swap and *swap(t,o)* actually performs the swap to take out t and insert o in the centroid set. Both operations between centroid t and non-centroid o are performed locally, involving both the members of the cluster of t and the neighbors of o as pictured in Figure 1. The group affected by each swap includes the cluster of t, which will be assigned to a new centroid because their centroid t is removed and the neighbors of o which will be re-assigned to o because of the cost reduction.

The set of members involved in *test-swap* and *swap* are largely overlapping for different swap targets and this fact is used to save resources by *combining* multiple *test-swap*s. Since there are fewer members per cluster in our domain, this combined message does not get very large. Additionally, multiple swaps can occur simultaneously only when the nodes affected by the swaps do overlap. If there are multiple overlapping swaps, the conflict will be resolved and only one swap will be selected.

After a finite number of swaps, the algorithm reaches a local optimum and terminates. Since we consider all nodes affected by the swap in and out of the cluster, the algorithm always calculates the exact change in the cost improving the cost each swap. Thus, it is guaranteed to terminate. The solution that this algorithm gives is different from Local search heuristics as the swap operation only occurs between the members of a cluster. Figure 2 provides an example showing how a sub-optimal solution will be found when only swaps within the cluster are done.

The Cluster-Swap algorithm has the benefit that the solution can be easily updated in a local manner after the addition of new nodes or the deletion of nodes in the network. This is because the addition and deletion impact can be determined locally by performing another iteration of *test-swap*s. If it is found that this will result in changes, the impact is then propagated by changing the centroids, resulting in the centroids in the neighborhood detecting the change and performing *test-swap*. Therefore, the propagation of impact is selective, making the update easier than other approaches where the impact always has to be considered globally.

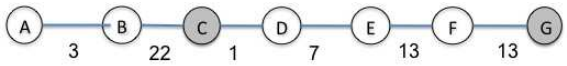


Figure 2. Example where Cluster-Swap does not lead to the same solution as Local search heuristics. The initial configuration is with two clusters of centroid C(Member A,C, D, E), and G(Member F). The cost of the initial configuration is 68(C:c(A,C)+c(B,C)+c(C,D)+c(C,E), G:c(F,G)). swap(G,A) improves the current solution, which is not performed by the Cluster-Swap. The cost benefit of swap (G,A) for A is 25 as it becomes a new centroid and cost is 0. For node G, the benefit is -34 as the new cost is 34, and cost before swapping is 0. The sum of the benefit from the swap $= 25 + 19 + 0 + 0 - 8 - 34 > 0$. Therefore Local search heuristics performs this swap and improves the cost.

2.1. Determining the error bound of Cluster-Swap

For the proof of an error bound, we use an approach similar to the one described for the Local search heuristics [1], applying k swaps between the centroid set T and the optimal solution O. We provide some terminology for proving the bound. Let $C(X)$ be the sum of the costs of all nodes on the graph given the centroid set X. $C_T(x)$ is the cost of x given the centroid set T and $c(x, y)$ is the cost between x and y. The proof is achieved in the following steps:

1. We create a mapping $M : T \rightarrow O$, from the solution T of the Cluster-Swap algorithm to optimal solution O. Such mapping leads to an element $o \in O$ appearing exactly once. To each $o \in O$, we map $t \in T$ such that $o \in N_T(t)$ where $N_T(t)$ is the set of members of the cluster of a centroid t given the centroid set T.

We cannot create a mapping $M : T \rightarrow O$ as in the paper of Local search heuristics. The mapping $S \rightarrow O$ is from the Local Search Heuristics solution $s \in S$ to optimal solution $o \in O$ that satisfies the constraint that $s \in S$ does not capture any element in O or capture only $o \in O$. This mapping is built based on the $swap(s, o)$ on S which increases the solution. However, the solution of Cluster-Swap algorithm does not guarantee this condition on the solution and may improve the cost in our solution. This fact violates the purpose of this mapping for proving the bound. For our purpose, any o should be mapped from the centroid t where $o \in N_T(t)$. However, this mapping does not hold the same constraint of capturing.

2. $C(T - t + o) \geq C(T)$ where $o = M^{-1}(t)$ as o is a member of $N_T(t)$, and swapping with a member of the cluster would not improve the solution.

3. Now we assign all members in $N_T(t) \cup N_O(o)$ to o. Consider the possible upper bound of the cost increase of each swap on $N_T(t) \cup N_O(o)$.

For $N_T(t) - N_O(o)$, since the closest centroid we can locate for $N_T(t)$ after the swap is o, we provide the upper bound of cost increase by assigning the members in

$N_T(t) - N_O(o)$ to o. The new cost for member x in $N_T(t)$ is bounded by $c(x,t) + c(t,o) = c_T(x) + c(t,o)$ as this cost does not exceed $c(x,o)$ by triangular inequality. Therefore, for each member in $N_T(t) - N_O(o)$, the cost increase is bounded by $c(t,o)$.

For members in $N_O(o)$, the cost of the node x before the $swap(t,o)$ is $C_T(x)$. The cost after swap when they are assigned to o is $c_O(x)$. For each member, the cost change is $C_T(x) - C_O(x)$.

4. For each swap, the cost increase is summarized as:

$$\Sigma_{x \in N_T(t) - N_O(o)} c(t,o) + \Sigma_{x \in N_O(o)} C_O(x) - C_T(x)$$
$$= c(t,o) * |N_T(t) - N_O(o)| + \Sigma_{x \in N_O(o)} C_O(x) - C_T(x)$$
$$\geq 0$$

5. For k swaps,

$$\Sigma_{\forall o \in O, t = M^{-1}(o)} (c(t,o) * |N(t) - N(o)|) + C_O - C_T \geq 0$$
$$C_T \leq C_O + \Sigma_{\forall o \in O, t = M^{-1}(o)} (c(t,o) * |N_T(t) - N_O(o)|)$$

However, we do not know the number of members of $N_T(t); N_O(o);$ or c(t,o) since we do not know o. Therefore, we calculate the maximum possible value that is $max[c(t,o) \times |N(t) - N(o)|]$. Since o a is member of $N(t)$, we can bound this value to $max(c_{max}(t) \times |N(t)|)$ for each t where $c_{max}(t)$ is the maximum cost between the centroid and non-centroid nodes within the cluster of t. Therefore, we can relate our solution using the maximum cost value multiplied by the number of members assuming each cluster does not contain more than a certain number of centroids in the optimal solution. Let $K_O(t)$ be the number of centroids in the optimal solution O, contained by the cluster of centroid t. We can bound our solution using $max_{t \in T}(K_O(t))$ and represent the equation as follows:

$$C_T \leq C_O + max_{t \in T}(K_O(t)) \times max_{t \in T}(c_{max}(t) * |N_T(t)|)$$

Our solution is bounded in relation to the maximum number of centroids of optimal solution in each cluster. Any cluster centroid t can be used in these swaps at most k times. The total sum for k swaps is bounded by $k(|N(t)| \times max_{x \in N(t)} c(t,x))$, which is at least maximum value of the cost of the entire graph making the bound useless. We can reason that $max(K_O(t))$ will remain lower than k, although we do not have a theoretical bound on $max_{t \in T}(K_O(t))$. This is because we are swapping with a node in the cluster, and thus there is a high chance that the swap with this centroid would result in a benefit unless the input nodes are extremely skewed and the algorithm starts from a poor random initial point. The experimental result also shows that $max(K_O(t))$ is much smaller than k in most cases.

2.2. Algorithm Details

The overview of the algorithm for a centroid is given in Algorithm 1.

[Initiation] The first step in the algorithm is to construct the initial clusters created by pre-determined or randomly chosen k centroid nodes that declare themselves as centroids by broadcasting the message to their neighbors. The message

```
members ← nil {member of its cluster}
tried ← nil {the members tried test-swap on}
if not initialized then
    send declare message to neighbors
    wait c × size { wait for the members to reply for time proportional to the size
    of the network}
end if
while type == centroid { until it becomes a non-centroid } do
    wait ← random(c × size)
    while wait >= 0 do
        process received messages
        if the benefit of test-swap is greater than 0 then
            send new centroid message to swap target
            send swap message to its neighbors
            type ← non − centroid
        end if
        wait ← wait − 1
    end whiletest-swap(tried, members)
    if not waiting for the replies, send test-swap message to its neighbors
end while
```

Algorithm 1: Function : *Cluster-Swap(k, type, neighborinfo)* **for the centroid**

of centroid declaration has a format <*declare-centroid, centroid id, cost*>.

Non-centroid nodes determine their membership in a cluster by selecting the known closest one upon receipt of these declaration messages, then notify their centroids by a message <*notify-membership, id*> and forward the declaration messages to their neighbors only when the source of this message becomes their own centroid. By forwarding the messages only from their own centroids, most nodes receive these declaration messages from only their centroids.

If any closer centroid is found by another declaration message, the receiver updates its membership and notifies the previous and new centroid nodes of its new membership by sending <*notify-membership-change, id*> and <*notify-membership, id*> .

For example, in Figure 2 node E registers itself as a member upon receiving <*declare-centroid, G, 26*>, then it sends to node G <*notify-membership, E*>. If later node E receives another declaration message <*declare-centroid, C, 8*> then node E changes the membership because there is a cost reduction from $26 < 8$. It notifies the membership update by sending <*notify-membership-change, E*> to centroid G and <*notify-membership, E*> to centroid C. Therefore, each node eventually becomes a member of the cluster of the closest centroid.

the number of messages may be reduced by waiting for an interval to get centroid declaration messages from neighbors, although this will increase the time for initialization.

```
if ∀x ∈members is in tried then
    state ←< stable > return
else
    starget ← a randomly selected non-centroid member in members not in tried
end if
state ←< testswap >
swapcost = −cost(src, starget) {cost change due to swap for the centroid}
for all i ∈ neighbor do
    send a message < testswap, src, starget, neighborcentroids, id >
end for
```

Algorithm 2: Function : test-swap($tried$**,** $members$**) : function by centroid to start test-swap**

[Test-Swap] *test-swap* is a message that is sent by centroids

```
newcentroid ← centroid with lowest cost among starget, neighborcentroids
if myswapsrc == src and swapid! = id
{same centroid sends again with new swap. It means the previous test-swap returned
negative result} then
    myswapid ← id {update the swap id }
    swapbenefit = cost − cost(newcentroid)
    if centroid == src||swapbenefit > 0 {if this node is a member of src}
    then
        send a message <test-swap-response, id, swapbenefit>
    end if
    for all i ∈ neighbor do
        send a message <testswap, src, starget, neighborcentroids, id>
    end for
else if id < myswapid then
    send a message < suppress, starget, src, id > {suppress the swap}
else if myswapid < id then
    send a message <suppress, starget, myswapsrc , swapid> {suppress the old
    swap}
    send a message <test-swap-response, id, swapbenefit>
end if
```

Algorithm 3: Function: process-test-swap(msg :<*src, starget, neighborcentroids, id*>**,** *myswapid, myswapsrc*) **: a function to process test-swap where myswapid, myswapsrc is info from a test-swap the nodes previously responded to**

```
if Response received from all of its neighbors then
    if testresult > 0 then
        for all i ∈ neighbor do
            send a message < swap, src, starget, id >
        end for
    else
        reset the timer to start test-swap
    end if
else
    testresult ← testresult + swapbenefit
end if
```

Algorithm 4: Function: process-test-swap-response (msg :<*test-swap-response, id, swapbenefit*>**)**

to calculate the cost of a potential swap. The format of *test-swap* is <*test-swap, swap id, source, target, [centroids in the neighborhood]*>.

If a centroid is contained in a cluster with at least one non-centroid node, the centroid chooses one of its members within the cluster and starts *test-swap* with a random interval [0, range] by sending messages to the all of their neighbors. This interval is in order to avoid conflicts between multiple swaps happening at the same time, where *range* is proportional to the number of nodes in the network.

Each *test-swap* message is forwarded while the potential swap affects the cost of the non-centroid node. The *test-swap(t,o)* is forwarded to $N(t) \cup N(o)$+ direct neighbors of $N(t) \cup N(o)$. Upon receiving the *test-swap* message, each non-centroid node calculates the benefit of swap, and determines whether to reply immediately or forward. The recipients of this message in $N(t)$ calculate the cost and forward the message to their neighbors regardless of the cost. The message will eventually reach the nodes outside $N(t)$ and they will return the messages with cost 0. On the other hand, the recipients of this message in $N(o)$ calculate the cost change and forward this message to its neighbors only if there is a cost decrease. If the swap only increases the cost, then it returns the message with 0 cost to the forwarder. If the forwarder gets replies from all of the neighbors, then it sends back the replies to its forwarder and eventually the centroid who started *swap-test* message will get the replies from all of its neighbors, and calculate the benefit by summing the results.

For example, as in Figure 2 suppose a node D who is a member of node C's cluster receives <*test-swap, 3456, C, B, [G]*>. Because the test message is from its own centroid, the node D must participate in the test, records the message, and forwards the message to its neighbors. It records the message to resolve the conflict between multiple swap messages which is explained later. When all replies for the test are returned to node D(here only from E), it calculates its own benefit, $benefit = c(C, D) - min(c(D, B), c(D, G))$, combines the result from all of its neighbors, and returns <*test-swap-response, 3456, C, D,* $\sum_{replies\ and\ its\ own}$ *benefit*> to the message's forwarder (the direct neighbor who sent it). Here, the benefit of E is $c(C, E) - c(E, G) = 8 - 26 = -18$. The benefit of D is -22 thus D sends a reply <*test-swap-response, 3456, C, B, -40*> to the node C. A slightly different action is taken when a node E who is not a member of node G receives a *test-swap* (G,F) message from G. Because $benefit <= 0$, it simply returns the response with 0 value and does not forward this message to a neighbor.

[**Neighborhood Centroid Information**] After a swap, the swap source is no longer a centroid. Thus, during *test-swap* the non-centroids calculate the cost based on the potential centroid set, and the centroid provides neighbor centroid information for cost calculation consistency. Additionally, if there is a change on the information during *test-swap*, the *test-swap* is terminated without any further effect.

[**Collision**] Also, there is a possibility of a conflict between multiple *test-swap* messages from different centroids, these conflicts must be resolved to avoid an incorrect test result. As stated before, the centroids try *test-swap* with some random interval in order to avoid these conflicts as much as possible, although having the interval does not eliminate the chance of conflicts. The collision inevitably happens because the centroids do not coordinate the time to start the swap. Instead, whenever a collision occurs, the swap with higher id (that is randomly generated for each *test-swap*) is chosen, and the other one is suppressed.

For example, from the previous example of a *test-swap*, the message from C will be forwarded to E as E is a member of the cluster. If the node E receives another *test-swap* message with *swap-id* 2487 from a different centroid G then it suppresses the *test-swap* with id 2487 since $2487 < 3456$ by directly sending the forwarder of *test-swap* a *suppress* message if the scheme is to suppress the smaller id. Since the source never swaps unless it gets all the replies from all of neighbors this *suppress* message will be delivered to its source before the swap happens.

[**Combining test-swap**] As in Figure 1, *test-swap* involves the members in the cluster and the nodes around the swap target. Therefore, large numbers of nodes on *test-swap* operations within the same cluster overlap and this fact is used to combine multiple *test-swap* operations to save communication resources. Multiple *test-swap*s can be done by in-

cluding multiple swap targets in one message and getting the calculated benefits from participating nodes in the *test-swap*. Each recipient of the combined message calculates the cost of each targets and returns the result on all targets in the message. Since multiple targets have different neighbors, the messages are propagated to slightly more nodes.

[Swap] If the centroid calculates the cost and there is a reduction, it determines to swap, and sends a *swap* message to its neighbors and swap target. This message is forwarded to all affected nodes, as in Figure 1. When a new centroid is elected by receiving a *new-centroid* message, it updates its type and copies the neighborhood centroid information, then forwards the *swap* message to its neighbors as well. The recipients of this message will notify the appropriate nodes of their membership change, if any.

[Termination] The algorithm terminates when no centroid can update its solution by swapping with other nodes in the same cluster. Once a node has tried every possible member of the cluster to swap, it stops *test-swap*s. The global termination point for the algorithm is when all the centroids agree to terminate because they stop generating*test-swap*s. There is no global termination mechanism implemented in our algorithm, however this decision problem is a common problem in the distributed environment for reaching a consensus, and there is ample literature on how to reach a consensus among k nodes in the network. Thus, the details are not provided in this work.

3. Empirical Evaluation

We present an empirical evaluation on synthetic data randomly created on an n-dimensional space with a size of 100 for each dimension. We focus on relatively small examples when compared to usual clustering domains since we model each node as a sensor or processor unit.

Table 1. Comparison between three algorithms used in the experimental section

	Neighbor-Swap	Cluster-Swap	Local Search Heuristics
Target of swap	direct neighbor	members of cluster	any non-centroids

We compare the Cluster-Swap algorithm with the Local search heuristics and the Neighbor-Swap algorithm, and for a limited number of cases an exhaustive search algorithm with the optimal solution. The difference among these algorithms is also provided in Table 1. We vary the number of points, centroids, and data dimensions. We assume the graph is not fully connected and the cost function is the distance between any two points(euclidean distance of any dimension). This assumption can be easily loosened by setting the cost function differently such as to the sum of the path to reflect the actual travel distance in the graph, or any other measure that also retains the triangular inequality. Our cost function is chosen for convenience.

The Cluster-Swap algorithm is also implemented in the distributed simulation environment Farm [10]. By implementing the Cluster-Swap algorithm in a distributed environment, we investigate additional measures: the number of communication messages, and time for convergence. In the Farm environment, "pulse" is used as the time unit and each action takes one pulse. Such actions include sending messages, receiving messages, and processing messages. Communication with any node in the network is sent within one pulse without any message loss.

3.1. Performance by Solution Quality

Tables 2 shows the maximum and average ratio of solution cost from Cluster-Swap and Local search heuristics. The results show that the solution by Cluster-Swap is close to the solution of Local search heuristics even though the search space of Cluster-Swap is more limited than Local search heuristics. In comparison to Neighbor-Swap, the Cluster-Swap algorithm performs closer to Local search heuristics than Neighbor-Swap algorithm. On some worst cases, the solution of Neighbor-Swap is 3 times larger than Local search heuristics, but the worst case ratio of Cluster-Swap is 1.49.

Table 3. Maximum Ratio of the solution of the Neighbor-Swap, Cluster-Swap and Local search heuristics compared to the optimal solution on 3 dimensions for 100 data points

Number of Centroids	Neighbor-Swap	Cluster-Swap	Local Search
4	3.104	1.081	1.000
8	3.414	1.000	1.000
12	2.587	1.829	1.412
16	4.203	1.636	1.636

Table 3 shows worst case ratio of the solution cost of the algorithms to the optimal solution on 3 dimensional data. The result shows that the Neighbor-Swap can lead to a poor local optimum and that the Cluster-Swap and Local search heuristics are both close to optimal even in the worst case.

3.2. Efficiency by Number of Swaps

We present efficiency results by considering the cost for *swap* and *test-swap* operations. The number of operations indicates how many times these routines are called, and the number of participants how much each operation costs in terms of both time and messages. The number of participants determines the runtime and the number of messages of each call since it indicates how much the information has to be propagated. Since *swap* and *test-swap* are dominant routines, the runtime and number of messages are proportional to the the number of operations and number of participants.

Table 4 shows the number of *test-swap* operations(The number of *swap* operations are dominated by *test-swap* by more than 10 times). The Local search heuristics requires the most operations as each centroid must *test-swap* with every node in the network, which leads to an explosion. For an uncombined *test-swap*, Neighbor-Swap remains much lower than Cluster-Swap. However, when the *test-swap* is combined for the members in the cluster, the two algorithms require a similar number of operations.

Because with *test-swap* messages combined the number of operations are similar both in Neighbor-Swap and

Table 2. Maximum and Average Ratio between the solution cost of Cluster-Swap(CS) and Local search heuristics(LSH) and Maximum Ration between the solution cost of Neighbor-Swap(NS) and Local search heuristics(LSH)

	Maximum/Average Ratio between CS and LSH						Maximum/Average Ratio between NS and LSH					
	2 dimensions			3 dimensions			2 dimensions			3 dimensions		
# centroids	100	200	300	100	200	300	100	200	300	100	200	300
4	1.137/1.029	1.072/1.017	1.010/1.001	1.073/1.010	1.378/1.061	1.151/1.023	1.745/1.197	3.007/1.326	2.378/1.261	1.336/1.124	1.545/1.155	1.602/1.155
8	1.125/1.043	1.236/1.045	1.068/1.009	1.107/1.026	1.058/1.008	1.078/1.029	1.372/1.154	2.408/1.496	1.759/1.412	1.276/1.157	1.702/1.352	2.130/1.381
12	1.293/1.064	1.052/1.022	1.076/1.014	1.183/1.063	1.081/1.032	1.123/1.027	1.616/1.233	1.973/1.310	1.725/1.406	1.904/1.402	1.461/1.298	1.648/1.453
16	1.163/1.069	1.094/1.045	1.072/1.039	1.486/1.134	1.067/1.033	1.052/1.021	2.411/1.267	2.194/1.367	1.874/1.307	2.000/1.376	2.253/1.570	1.957/1.394

Table 4. Efficiency of Algorithms: number of *test-swap* operations until the algorithms reach the local optima on 3 dimension, number of *test-swap* operations if combined in one message, and number of participants on each combined *test-swap*

	number of *test-swap* if not combined									number of *test-swap* if combined						number of participants if combined					
	Neighbor-Swap			Cluster-Swap			Local Search Heuristics			Neighbor-Swap			Cluster-Swap			Neighbor-Swap			Cluster-Swap		
# centroids	100	200	300	100	200	300	100	200	300	100	200	300	100	200	300	100	200	300	100	200	300
4	83.6	132.7	102.0	363.7	647.2	1062.3	1064.2	2507.8	3892.4	34.2	61.4	45.1	31.2	31.6	34.0	32.0	61.7	95.4	55.1	132.3	236.8
8	171.7	241.2	255.0	438.9	1031.4	1708.3	2087.8	4725.7	7131.5	76.4	106.9	119.9	60.8	80.9	91.3	16.1	34.5	60.7	28.2	73.1	128.5
12	194.2	281.8	256.8	517.6	1058.8	1786.2	2842.0	7409.5	10002.9	87.2	126.5	116.0	94.2	104.9	125.0	11.7	24.2	38.6	18.2	46.3	86.1
16	248.9	332.7	318.8	406.5	1485.1	1988.8	4369.9	9939.0	15424.5	119.0	151.1	148.7	147.9	178.4	168.4	9.8	19.2	27.2	12.8	33.0	63.5

Cluster-Swap, the number of participants determines the total cost. On each combined *test-swap* of Cluster-Swap, less than or similar to twice the number of participants of Neighbor-Swap participates. Considering that Neighbor-Swap quickly reaches a sub-optimum and terminates, the efficiency of Cluster-Swap is reasonable. Although the message size in Cluster-Swap is bigger, it only contains an array of size of a cluster(around $6-75$ in our experiments on average). Additionally, the messages travel only one hop to only nodes' direct neighbors. We thus expect the message costs in both algorithms to be similar. Therefore, Cluster-Swap has a comparable efficiency in the runtime and communication load with Neighbor-Swap.

3.3. Solution Quality in terms of Error Bound

Table 5. Average of maximum cluster value for the cost bound for 3 dimensional data on 100 data points and average of maximum K where K is the number of centroids from the optimal solution within a cluster

Number of centroids	4	8	12	16
Maximum cost	4965.0	1921.3	1233.5	845.4
Optimal cost	(5850.2)	(2836.9)	(1915.3)	(1262.4)
max K	2.2	2.7	3.1	3.4

The Cluster-Swap algorithm's performance bound depends on the *maxc* of a cluster where we define *maxc* as $max(|N(t)| \times max_{x \in N(t)} c(t,x))$. *maxc*. The solution bound is calculated as *maxc* multiplied by $max_t(K_O(t)) = K_{max}$ where $K_O(t)$ is the number of centroids from the optimal solution O in cluster of the centroid t. As shown in Table 5, K_{max} is on average is around 3 in our experimental result and the bound for Cluster-Swap is therefore less than triple the optimal cost(making it 3-approximate). Although maximum $K_O(t)$ can be very large for the extreme cases when Cluster-Swap's solution is very different from the optimal solution due to the poor initial centroid sets, the error bound is expected to be held in a more common cases. This error bound is good, considering that the centralized k-median algorithm such as in Local search heuristics is only 5-approximate and other streaming or distributed algorithms are not better than 5-approximate.

3.4. Communication and Convergence

Table 6. Average number of received messages and pulse taken to converge by Cluster-Swap with 50 nodes in the network

Number of centroids	4	8	12	16
# messages	5022.2	2689.6	2064.4	1567.6
# pulse	2849.8	1194	1250.8	1136.4

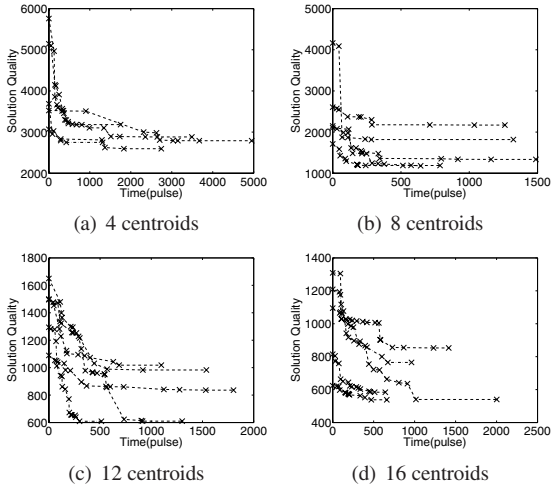
In Table 6, we measure the number of messages needed to complete the task of Cluster-Swap with uncombined *test-swap* in the network with 50 nodes. Since we assume no message loss, the number of sent message is identical. On average each node communicate $60-200$ messages. When the centroid set is smaller the cluster gets bigger, requiring more messages. Most messages are *test-swap* and *swap* messages between a node and its direct neighbors so their cost is minimal. Considering the number of *test-swap* operation each node is involved in, other control messages do not change the communication cost much.

Table 6 also shows the result of pulse required for convergence. Since each *test-swap* and *swap* operation takes twice the maximum length of path in a cluster, considering the number of these two operations, the algorithm does not lose too much efficiency because of the suppressed *test-swap*s.

Figure 3 shows the change of cost during several runs of Cluster-Swap algorithm with 50 nodes. It shows that the time to reach the cost closest to the local optima is pretty small in comparison to the total time until convergence and it is not necessary to wait until the algorithm actually converges. After a short time, the solution stops significantly improving. Additionally, when there are fewer members in the cluster, convergence detection is faster and it terminates quickly as in the case with 16 centroids.

4. Conclusion

In this paper, we provide a mechanism to solve the k-median algorithm in a distributed environment where the



(a) 4 centroids (b) 8 centroids

(c) 12 centroids (d) 16 centroids

Figure 3. The change of cost during Cluster-Swap algorithm performed on 50 nodes with 4,8,12 and 16 centroids

exact locations of nodes are not known. The Cluster-Swap algorithm, which only uses a swap operation between a centroid and a non-centroid node within the same cluster, can be more efficient in comparison to Neighbor-Swap and Local search heuristics.

The algorithm has a performance bound proportional to the maximum cost of a member in a cluster and the number of members in the same cluster given the assumption on K_{max}. K_{max} is a measure of the maximum number of centroids from the optimal solution within one cluster in the solution from Cluster-Swap, and it is experimentally shown that if the initial centroid set is not extremely skewed, it remains much lower than k - the number of centroids.

Experimental results show that Cluster-Swap achieves a good quality solution with moderate efficiency. Neighbor-Swap reaches a suboptimal solution quickly only requiring a small number of messages since the search space is very limited. However, Cluster-Swap converges to a solution as good as the Local search heuristics with less than or similar to twice the messages of Neighbor-Swap.

As future work, to explore more about the actual communication cost considering the size and number of messages and the effect of suppressed swap operations due to conflicts on overall efficiency of the algorithm would be interesting to compare the algorithms in more detail.

Acknowledgement

This work was supported in part by the Engineering Research Centers Program of the National Science Foundation under NSF Cooperative Agreement No. EEC-0313747. Any Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

References

[1] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristic for k-median and facility location problems. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 21–29, New York, NY, USA, 2001. ACM.

[2] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD Conference*, pages 28–39, 2003.

[3] M. Charikar. Better streaming algorithms for clustering problems. In *In Proc. of 35th ACM Symposium on Theory of Computing (STOC*, pages 30–39, 2003.

[4] M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. In *In Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 1–10, 1999.

[5] A. Das and D. Kempe. Sensor selection for minimizing worst-case prediction error. In *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*, pages 97–108, Washington, DC, USA, 2008. IEEE Computer Society.

[6] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *International Conference on Very Large Data Bases*, pages 588–599, 2004.

[7] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. In *IEEE Transactions on Knowledge and Data Engineering*, volume 15, pages 515–528, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[8] S. Har-Peled and S. Mazumdar. Coresets for k-means and k-median clustering and their applications. In *STOC*, pages 291–300, 2004.

[9] T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, and B. Krogh. Energy-efficient surveillance system using wireless sensor networks. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 270–283, New York, NY, USA, 2004. ACM.

[10] B. Horling, R. Mailler, and V. Lesser. Farm: A Scalable Environment for Multi-Agent Development and Evaluation. In A. G. C. Lucena, J. C. A. Romanovsky, and P. Alencar, editors, *Advances in Software Engineering for Multi-Agent Systems*, pages 220–237. Springer-Verlag, Berlin, February 2004.

[11] S. G. Kolliopoulos and S. Rao. A nearly linear-time approximation scheme for the euclidean k-median problem. In *SIAM J. Comput.*, volume 37, pages 757–782, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[12] A. Meliou, D. Chu, J. Hellerstein, C. Guestrin, and W. Hong. Data gathering tours in sensor networks. In *IPSN '06: Proceedings of the 5th international conference on Information processing in sensor networks*, pages 43–50, New York, NY, USA, 2006. ACM.

[13] J. Robinson, M. Uysal, R. Swaminathan, and E. W. Knightly. Adding capacity points to a wireless mesh network using local search. In *INFOCOM*, pages 1247–1255, 2008.

[14] J. L. Williams, J. W. Fisher, and A. S. Willsky. Approximate dynamic programming for communication-constrained sensor network management. *Signal Processing, IEEE Transactions on*, 55(8):4300–4311, 2007.