# Applying Electromagnetic Field Theory Concepts to Clustering with Constraints

Huseyin Hakkoymaz[1], Georgios Chatzimilioudis[1], Dimitrios Gunopulos[1,2], and Heikki Mannila[3]

[1] Dept. of Computer Science, University of California, Riverside, CA, 92521, USA
{huseyin,gchatzim,dg}@cs.ucr.edu
[2] Dept. of Informatics and Telecommunications, Univ. of Athens, Greece
[3] HIIT, Helsinki University of Technology and University of Helsinki, Finland
heikki.mannila@cs.helsinki.fi

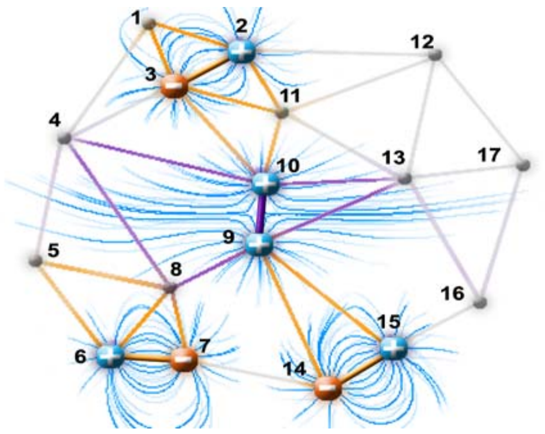
**Abstract.** This work shows how concepts from the electromagnetic field theory can be efficiently used in clustering with constraints. The proposed framework transforms vector data into a fully connected graph, or just works straight on the given graph data. User constraints are represented by electromagnetic fields that affect the weight of the graph's edges. A clustering algorithm is then applied on the adjusted graph, using *k*-distinct shortest paths as the distance measure. Our framework provides better accuracy compared to MPCK-Means, SS-Kernel-KMeans and Kmeans+Diagonal Metric even when very few constraints are used, significantly improves clustering performance on some datasets that other methods fail to partition successfully, and can cluster both vector and graph datasets. All these advantages are demonstrated through thorough experimental evaluation.

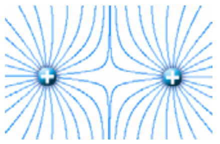
**Keywords:** Data Clustering, User Constraints, Electromagnetic Field Theory.

## 1 Introduction

The goal of clustering is to provide useful information by organizing data into groups (referred to as clusters). The use of labeled data is often important for the success of the clustering process and for the evaluation of the clustering accuracy. Consequently, learning approaches which use both labeled and unlabeled data have attracted the interest of the research community [3, 4, 6, 12, 20]. Such approaches incorporate user knowledge in the clustering technique, thus improving the clustering result. There are several ways to incorporate user knowledge in the clustering process. In this work we focus on the Clustering with User Constraints paradigm, where the user specifies constraints on groups of objects (typically pairs), and the goal is to produce a clustering that satisfies these constraints as much as possible.

In this paper, we present a novel way of applying user constraints for clustering; our approach is inspired by the Electromagnetic Field Theory in physics. We transform the dataset into a graph by linking *k*-nearest neighbors for each instance in the dataset, if the data are given as vectors. If the input is a distance matrix, no transformation is needed. Must-link and cannot-link constraints are then expressed naturally





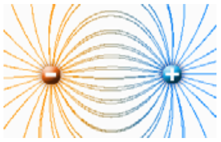


**Fig. 1.** The simulation of an EMF in a graph. (a)Like charges induces repulsive properties while (b) opposite charges induces attractive properties of other objects.

as magnetic fields between the nodes that are involved in the constraint. These fields impact edge weights based on the alignment of each edge compared to the magnetic field and its distance to the constraint axis. Using a graph representation yields the advantage that the edge weights can be adjusted without limitations, in contrast to Euclidean space where pairwise distances need to satisfy the triangle inequality. We exploit this liberty through a probabilistic model based on the nature of the constraint edges.

**Contributions:** We present a framework, called *EMC*, for clustering with constraints inspired by Electromagnetic Field Theory. The framework is general and assumes only that a distance function is available. The distance function does not need to satisfy the triangle inequality. If vector data is given, we transform it into a graph with minimal information loss; else if graph data is given we work directly on the given graph. We adjust the edge weights according to their proximity to the constraints in a probabilistic manner and run any clustering algorithm compatible with graphs. In our study, we use the K-Medoids algorithm [10], since it is less sensitive to outliers. For the distance metric to be used by the clustering algorithm, we propose *k simple-and-distinct shortest paths,* described in section 4.3. It allows for accurate clustering even with small number of constraints. Our method avoids the use of an objective function that strictly adheres to the given constraints, or a distance metric that is weighted according to the constraints and is applied globally.

Our experimental results confirm that our approach outperforms in accuracy current techniques including MPCK-Means [4], SS-Kernel-KMeans [12] and KMeans+ Diagonal Metric [20] even using very few constraints. Also, it can work on both vector and graph datasets.

## 2 Related Work

COP-KMeans [18] uses must-link and cannot-link constraints in an objective function to avoid incorrect assignment of data instances. It performs hard constrained clustering and computes the transitive closure of the constraints, thus suffering greatly from noisy constraints sensitivity. On the contrary, Xing et al. [20] proposed a distance metric learning algorithm which places a distance metric over the input space with the intention of assigning small distances between similar pairs. Distance metric learning aims to satisfy the maximum number of constraints by specifying different weights for different axes. The RCA algorithm [2] learns a Mahalanobis distance metric by using only must-link constraints. However, both approaches find one global metric which must be applied to all clusters.

MPCK-KMeans [4] integrates the strengths of both metric-based and constraint-based approaches in a principal manner. It learns individual distance-metrics for each cluster utilizing both unlabeled data and constraints. This allows different clusters to define their own space and have arbitrary boundaries. Its drawback: it suffers from noisy constraints.

HMRF-KMeans [3] is a probabilistic framework based on Hidden Markov Random Fields. Basu et al. have taken advantage of the available constraints in several ways. First, they estimate initial centroids using the constraints as the initialization step is crucial to accuracy of the KMeans algorithm. The model integrates constraint-based and distance-based approaches to maximize the joint likelihood of data and constraints while penalizing violated constraints. One weakness of the method is that it is applicable only to vector data, like all of the other mentioned so far.

Kulis et al. [12] extended HMRF-KMeans to a kernel-based framework which can handle both vector-based and graph-based data. They have established a connection between unweighted Kernel-KMeans, HMRF-KMeans and penalties for violated constraints. SS-Kernel-KMeans optimizes the kernel by preprocessing the similarity matrix with must-link and cannot-link constraints. Kernel methods are sensitive to the manual selection of the kernel's parameters. Yan and Domeniconi [22] proposed an adaptive method that estimates the optimal parameters. Our method adjusts edge weights more intuitively since it considers edges nearby the constraints as well, and this leads better performance. This model is based on the homophily concept [8], – nearby nodes tend to share same characteristics. Although the attraction/repulsion is practiced in unsupervised clustering [15], exploiting their use in semi-supervised clustering via the concept is completely new to our best knowledge.

Klein et al. [23] has based his algorithm on the same *homophily* intuition as our work, but can be applied only to vector data. Also, our method deals in a more comprehensive way with constraint satisfaction and triangular inequality restrictions.

## 3 Magnetically Affected Paths (MAP)

In this section, we describe the basic idea behind our approach in an intuitive way. We want to cluster a dataset of $n$ points. We assume that a function defining the distance between any two points and a set of user defined constraints are given. Our goal is to cluster the points so that the constraints are satisfied as much as possible. We make no assumptions on the distance measure, but we assume that the constraints are on pairs of points, either Must-Link (the two points must be in the same cluster) or Cannot-Link (the two points cannot be in the same cluster). We use the user-defined constraints to “stretch” the space around the object that are accumulated around the constraints. Since we are dealing with graphs, our goal is to increase the weight of the paths connecting the objects that are in the neighborhood of a cannot-link constraint, and to decrease the weight of the paths connecting the objects that are in the neighborhood of a must-link constraint.

We realize the idea via the interesting analogy between electromagnetic field theory and graphs. We focus on the graph representation of a given dataset and assume that this graph has the characteristics of an electromagnetic field (EMF). In physics, an electric field is the property of the space in the vicinity of electric charges or in the presence of a time-varying magnetic field. The charges produce an electric field in space. This electric field exerts a force on other charged objects [14]. Opposite charged objects induce attractive properties, whereas like charged objects induce repulsive properties. To simulate these characteristics, we add charges to the nodes that take part in a pair-wise constraint. For must-link constraints, we add opposite charges to the node pair. The generated magnetic field decreases the weight of the affected edges. Cannot-link constraints (like charges) increase the weights of affected edge. The situation is illustrated in Figure 1, where must-link constraints are {*e(2,3), e(6,7), e(14,15)*} and the cannot-link constraints are {*e(9,10)*}.

We explore imaginary magnetic fields surrounding the pair of charged nodes. We check the nearby edges in the graph and identify the graph edges that are influenced by the magnetic field. Then, we reduce the weight of the affected edge or escalate it according to the constraint type. The magnitude of readjustment depends on the distance of the edge in regard to the magnetic field and its alignment in the field.

Before we proceed to the MAP, some definitions are necessary:

- *Constraint axis.* The straight line between the pair of nodes involved in a constraint
- *Reduction ratio rRatio(u,v)* is the decrement amount in edge weight w(u,v) due to a must-link constraint.
- *Escalation ratio eRatio(u,v)* is the increment amount in edge weight w(u,v) due to a cannot-link constraint.
- *Vertical distance vd(u,v).* The average distance of edge e(u,v) to the constraint axis.
- *Horizontal distance hd(u,v)* is defined as distance of edge e(u,v) to the mid-point of the *constraint axis.*

The effect of an EMF decreases as we get further away from the constraint axis (vertical distance). For must-link constraints, the horizontal distance has no effect on the reduction ratio. Cannot-link constraints utilize the horizontal distance of an edge to determine its probability of being in the separation region of two clusters. Intuitively, *the closer a regular edge e(u,v) is to the mid-point of a negative edge constraint, the higher the probability of it being an inter-cluster edge.* Based on this intuition, we apply the highest penalty to the edges overlapping with the mid-point of a constraint axis. The penalty is reduced as we go further away from the mid-point.

To summarize, MAP increases or decreases the weights of regular edges based on a probabilistic approach. Even though it classifies some of the edges incorrectly, overall re-adjustment of edge weights defines better distances in the graph domain.

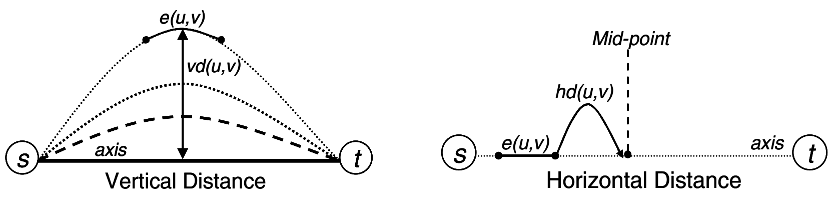


**Fig. 2.** Horizontal and vertical distances relative to the constraint axis (*s* and *t* are two nodes of a constraint)

## 4 EMC Framework

Here we describe our clustering framework that unifies MAP and clustering algorithms. The algorithm uses 3 steps to apply the MAP concept and cluster a given dataset: 1) *Graph Construction:* If given a vector-based dataset, it is converted into a graph by connecting *k*-nearest-neighbors. Must-link and cannot-link constraints are addressed as same or opposite charged nodes respectively. 2) *Weight Adjustment:* Identifies the edges that are affected by the given constraints and adjusts the edge weights accordingly. 3) *Clustering:* Runs an appropriate clustering algorithm to partition the adjusted graph.

### 4.1 Graph Construction

If the input is not a graph but a vector of data points *Ds*, our framework builds a graph reflecting the data with minimal loss of information. We list *k* nearest neighbors $L_i$ for each object $x_i \in Ds$, according to their Euclidean distance, and add an edge between $x_i$ and each $v \in L_i$. We assign the Euclidean distance $\|x_i - v\|_2$ as the edge weight of $e(x_i,v)$. *k* can be easily estimated from the graph size. Experiments show that *k* should be proportional to the dataset size $|Ds|$ for better accuracy.

We take all node pairs $(s_i,t_i)$ involved in some constraint and charge $s_i$ and $t_i$ so that the force between them is equal to $\|s_i - t_i\|_2$. Remember, opposite charged pair of nodes create an attractive force (must-link constraint), whereas same charged pair of nodes create a repulsive force (cannot-link constraint).

**Disconnected Components.** Even if we set *k* to its optimal value, disconnected components might still exist. We identify all disconnected subgraphs, explore *k* nearest neighbors between subgraphs, and add an edge for each nearest neighbor connecting the disconnected components. This approach is similar to [12].

### 4.2 Weight Adjustment Algorithm

The weight adjustment phase applies the MAP concept to the graph in order to increase or decrease the edge weights. For each constraint $c(s,t)$ , we extract a list *L* of edges $e(u,v) \in E$ which are affected by the constraint. Affected edges are labeled according to the following definition:

DEFINITION. *If an edge e(u,v) is in between nodes s and t of constraint c(s,t) and is not perpendicular to the constraint axis, then it is affected by constraint c(s,t).*

In our model, perpendicularity and betweenness is defined based on hop-count distance to the constraint pair nodes *s* and *t*. We run two breadth-first search algorithms starting at *s* and *t* separately and for each node we store entries *hc(u,s)* and *hc(u,t),* the hop-count distance to *s* and to the *t* respectively. For a given edge *e(u,v),* we check the hop-count entries of *u* and *v* to see whether the edge is affected by a constraint or not. Perpendicularity and betweenness is defined as the inverse behavior on *hc(u,s), hc(u,t)* and *hc(v,s), hc(v,t)* values for nodes *u* and *v*. In other words, if $hc(u,s)>hc(v,s)$ and $hc(u,t)<hc(v,t)$ or vise versa, then the edge is perpendicular to the constraint axis and between nodes *s* and *t*.

Once we identify the affected edges, we compute the escalation ratio for the cannot-link constraints or reduction ratio for the must-link constraints. In line with the main idea, we expect the effective escalation/reduction ratio on an affected edge to decrease as we get away from the constraint axis (vertical distance). For a cannot-link constraint, we expect an inversely proportional weight increase in regard to the distance of the edge to the mid-point of the constraint axis. To express this, we use a method similar to the validation process. Instead of hop counts, we find the shortest path distance to all edges starting at node *s* and *t*. Shortest path *dist(u,v)* is the sum of the weights of all edges that compose the shortest path. We compute the reduction ratio of *e(u,v)* for must-link constraints as follows:

$$rRatio(u,v) = norm(\frac{q_r}{r}) \tag{1}$$

Here, $q_r$ is the weight for the must-link constraint and $r=(dist(u,s) + dist(u,t) + dist(v,s) + dist(v,t)) \,/\, w(s,t)$ which is the dispersion ratio from the constraint *c(s,t). norm()* is the normalizing function. To prevent extremely low values, normalization function maps the output to a higher interval.

Similarly, we can write the following equation to compute the escalation ratio due to a cannot-link constraint:

$$eRatio(u,v) = norm(\frac{q_e}{r \cdot \Delta}) \tag{2}$$

where $q_e$ is the weight of the cannot-link constraint, $\Delta=(|dist(u,s)-dist(u,t)|+|dist(v,s) - dist(v,t)|)/w(s,t)+c,$ which is the average distance approximation function to the mid-point of the constraint axis to reflect the effect of horizontal distance as seen in Figure 2. The value of $|dist(u,s)-dist(u,t)|$ becomes zero if node u has equal distances to the s and t. We add a constant value $c=1$ to the Δ so that in this case, no penalty is applied to *eRatio(u,v).* If Δ increases, the effect of *eRatio(u,v)* reduces gradually.

After applying all constraints, we approximate the overall ratio of edge *e(u,v)* as:

$$tRatio(u,v) = \frac{\sum_{i=1}^{|C|} eRatio_i(u,v)}{|C|} - \frac{\sum_{j=1}^{|M|} rRatio_j(u,v)}{|M|} \tag{3}$$

In the last step, we adjust the edge weight as follows:

$$w_{new}(u,v) = w(u,v) \cdot \alpha^{tRatio(u,v)} \tag{4}$$

```
Algorithm. Weight_Adjustment_Algorithm
Input: G(V,E): graph with constraints
Output: G' (V,E'): graph with adjusted edge weights
        P: proximity matrix

1. For each constraint $c(s,t)$
    a. Run breath-first search algorithm starting at node $s$ and $t$;
       and record hop-counts for each node $v_i \in V$
    b. Run single shortest path algorithm starting at node $s$ and $t$;
       and record shortest path distances for each node $v_i \in V$
    c. Identify affected edges using hop-counts and put them into list L
    d. Compute escalation/reduction ratio for each affected edge $e(u,v) \in L$
2. For each edge $e(u,v) \in E$
    a. Calculate overall ratio using following formula
       $tRatio(u,v) = \frac{\sum_{i=1}^{|C|} eRatio_i(u,v)}{|C|} - \frac{\sum_{j=1}^{|M|} rRatio_j(u,v)}{|M|}$
    b. Apply overall ratio to the edge weight $w_{new}(u,v) = w(u,v) \cdot \alpha^{tRatio(u,v)}$
3. For each node pair (v, u), calculate the distance $D(u,v) = \left( \sum_{q=1}^{k} \frac{1}{dist_q(u,v)} \right)^{-1}$ .
```

**Fig. 3.** Pseudo-code of Weight Adjustment Algorithm

Empirically, we have observed that $1<\alpha<2$ is a good interval for the adjustment of the edge weights. It is obvious that if cannot-link constraints are dominant upon the must-link ones, then the edge weight increases. Note that vertical and horizontal distances are used just to compute the adjustment ratio and are not used in any way in the clustering process.

After adjusting all the edge weights, we have our final graph and the algorithm uses this graph to extracts distance matrix *D*. The distance between two node pairs is defined using *k*-shortest paths distance. Remember that *k* was the number of neighbors used in order to transform vector data into a graph representation. For vector data, we use the same *k* for the number of shortest paths since this is the maximum out-links number of an edge. Using multiple shortest paths as a distance between two nodes, works in practice better than a single shortest path. Involving more paths in the calculation of the distance involves also more constraints, which yields better accuracy even for a small number of constraints due to the homophily phenomenon. A naïve approach for k-shortest paths is using the Dijkstra's algorithm to discover *k* most significant paths one by one.

Each constraint affects the graph locally. Thus, we can focus on subgraphs that capture the relevant information we need rather than whole graph. We have adopted the idea of very small-connection subgraphs from [11]. The weight adjustment step asymptotically takes $O((|M| + |C|)(|E| + |V| \log|V|))$ time, but in practice it is much faster. We explore k-shortest path for every node pairs. Extracting the proximity table requires O($k$ |$V$|$^2$ (|$E$| + |$V$| log|$V$|)) time. Thus, the weight adjustment phase takes overall O($k$ |$V$|$^2$ (|$E$| + |$V$| log|$V$|)).

## 4.3 Optimization for K-SD Shortest Path Algorithm

Finding the $k$-shortest simple paths for every pair of nodes is the bottleneck for the efficiency of our algorithm. Even though there is more than one $k$-shortest paths definition in the literature [5], we focus on $k$ simple and distinct shortest paths in which no loop is allowed, i.e. all vertices on a path are distinct and no two paths share the same edge for a given source and destination pair.

We extend the single shortest path Dijkstra algorithm to k-shortest paths at a reasonable cost and refer to it as K-SD shortest path algorithm. The new algorithm is asymptotically only $k$ times slower than the Dijkstra algorithm for one path. For each node, we define $k$ entries to handle each $l^{th}$ path passing through, where $1 \leq l \leq k$. We initialize all entries to $\infty$ except the entries of source node $s$ and nodes adjacent to the source. We set all $s$ entries to 0. We assign monotonically increasing path labels $i$ to each node $u$ adjacent to $s$ ensuring that each path is rooted at a different edge outgoing $s$. We update the $l^{th}$ entry of each node $u$ to edge weight $w(s,u)$ and the parents of $l^{th}$ entry of the nodes to source node $s$ where $l$ is the path label pertaining to each adjacent node. Then, we initialize a minimum priority queue $Q$ that contains all path

```
Algorithm. K-SD_Shortest_Path_Algorithm
    Input: G'(V,E): graph with adjusted edge weights
               s: source node
               k: number of shortest path
    Output: P: Distance matrix
1. Assign $k$ path entries for each node such as $p_i(s,u)$
2. Initialize $p_i(s,u).length \leftarrow +\infty$ for each node $u \neq s$
3. For each node $u$ adjacent to $s$
   a. Assign a monotonically increasing path id $i$ to $u$
   b. Set $p_i(s,u).dist \leftarrow w(s,u)$ and $parent_i(u) \leftarrow s$
4. Let a min priority queue $Q$ contain all path entries
5. while $Q$ is not empty do
   a. Extract path entry $pe \leftarrow Q$.removeMin()
   b. Let $i \leftarrow pe.pathID$ and $v \leftarrow pe.node$
   c. Lock $parent_i(v)$ to prevent updates for $v$
   d. for each node $u$ adjacent to $v$
      i. if $v$ is locked for $u$, then continue
      ii. if $p_i(s,v).length + w(v,u) < p_i(s,u).length$, then
             $p_i(s,u).length \leftarrow p_i(s,v).length + w(v,u)$
             $parent_i(u) \leftarrow v$
             Update the value of $p_i(s,u)$ in queue $Q$
```

**Fig. 4.** Pseudo-code of K-SD Shortest Path Algorithm

entries. The rest of the algorithm works very similar to Dijkstra's shortest path algorithm except for a few additional restrictions in the relaxation routine.

When we remove the minimum entry from $Q$, we lock the parent of the entry in order to prevent further updates from this parent for other path entries, using bitmaps with each bit assigned to one neighbor node. In the relaxation procedure, first we check whether the current node is locked for destination node. If it is not, the algorithm allows it to relax the destination from current node. Otherwise, we simply proceed to the next available neighbor node. Another rule is that $l^{th}$ entry of path entries at a node can be updated only by the $l^{th}$ path entry of the parent node. This limitation is necessary in order to force all paths to follow a different set of edges to the destination. We repeat the relaxation process until all path entries at every single node are updated and no more entries reside in the queue (Figure 4).

THEOREM 1. *K-SD Shortest Path algorithm identifies k simple and distinct shortest paths from a given source node to all other nodes in* $O(k^2 |V| \log|V|)$ *time.*

**Proof:** We assume the worst case scenario where we find all shortest path available, giving us $O(k\ |V|)$ entries in the queue. For each entry, we check all neighbor nodes and based on $k$-shortest path definition and our graph construction algorithm, we may have at most $k$ neighbors. We may update the keys of all neighbors at each relaxation step, requiring $O(k \log|V|)$ time. Therefore, our algorithm takes overall $O(k^2 |V| \log|V|)$ time to find all k simple and distinct shortest paths from a given source node s to all other nodes.

## 4.4 Multi-level Approach for Extracting Similarity Matrix

Even with the K-SD shortest path algorithm from the previous section, it takes $O(k^2 \cdot |V|^2 \cdot \log|V|)$ to compute all pairs, which is inefficient for large datasets. Many state-of-the-art methods deal with this problem by using a multilevel approach. Our strategy is to partition the graph into smaller pieces, compute distance matrices for each partition, and correlate them via hubs to obtain the global solution [17].

We partition the graph into $p$ equally sized subgraphs using METIS with the Kernighan-Lin objective and find local K-SD shortest path distances for each partition. Let $D_i$ be the distance matrix for partition $i$. To establish a relationship between partitions and be able to estimate the global distances we use the hub concept discussed in [21]. The vertices that reside on the cut and bridge different clusters are considered hubs. We are interested only in high quality hubs, which have high degree and their edges are balanced amongst the different partitions it bridges. Unlike the hubs in SCAN, we assume hubs are special vertices belonging to all partitions that it bridges, as shown in Fig. 5. If needed, we add new edges to hubs to maintain the $k$-neighborhood.

Let $H$ be the set of all hubs in the graph, $H_i$ be the set of hubs in partition $i$, and $m$ be the size of $H$. Notice that $m<<n$, which makes a fast correlation possible. We already have computed the distance between nodes in partition $i$ and all hubs in $H_i$. Next, we find the pair-wise distances between hubs $H$ so that we can compute the distance of two elements that belong to two different partitions. Since $H_i$ hubs are also elements of another partition by definition, we get a fully connected graph of hubs, where edges represent the distance between two hubs. By running

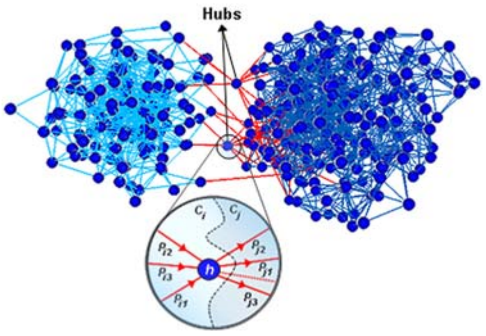


**Fig. 5.** The *Ionosphere* graph with two partitions and hubs connecting them, and the correlation of same paths for clusters $C_i$ and $C_j$

Floyd-Warshall on this imaginary graph, we get a distance matrix $S$ of all hubs in the graph. The matrix $S$ will serve as an approximation of the KSD-shortest path distance between two nodes of different clusters.

For example, assume we have performed all the preparation for extracting the global distances in an efficient way. Let $s$ be the source node, $t$ be the destination node, and $S(h_i,h_j)$ be the distance between hubs $h_i$ and $h_j$. Then, the KSD-shortest path problem can be expressed as the following optimization problem:

$$\begin{gathered} D(s,t) = min\{D_i(s,h_a) + S(h_a,h_b) + D_j(h_b,t)\} \\ \text{where } \forall h_a \in H_i \text{ and} \forall h_b \in H_j\} \end{gathered} \tag{5}$$

We have the distances between $s$ and all $H_i$ elements in matrix $D_i$. We compute the distances from $s$ to the other hubs in set $(H-H_i)$ through $H_i$ using the $S$ matrix. Each destination node checks only the hubs of its own partition in $H_j$ to relax the shortest path distance to node $s$. It takes $O(|H_i|\ (|H| - |H_i|) + |V - V_i|\ |H_j|)$ to find $k$-shortest paths from $s$ to all other nodes. Note that, $|V_i| \approx |V|\ /\ p$ and $|H_i| \approx |H|\ /\ p$. With this partitioning concept, our algorithm (*EMC*) has an overall time complexity of $O\left(\frac{|V|^2 k^2 \log(|V|)}{p} + |H|^3 + \frac{|V| \cdot |H| \cdot (|V| + |H|)}{p}\right)$. If $p=|V|$, it finds only single shortest paths. In practice, this approach runs hundreds times faster compared to the naïve approach.

## 4.5 Clustering Algorithm

The framework allows us to use any graph-compatible clustering algorithm. The success of clustering essentially depends on the compatibility of the dataset and the clustering algorithm. Thus, the choice of the algorithm must be made cautiously.

We implement the K-Medoids algorithm, which utilizes the similarity matrix and can be applied on both vector and the graph data. The initial centroids play a significant role for the final clustering, so instead of random initialization, we take advantage of the given constraints. We take the transitive closure of the must-link constraints and define groups of nodes, which have to be clustered together. At this point, each group represents a set. We merge the closest two sets until we have $K$ sets remaining. Eventually, we have $K$ disconnected sets formed by constraints, which we use to initialize the medoids.

The algorithm starts by assigning every point $x_i \in Ds$ to the cluster that minimizes the distance between $x_i$ and $\boldsymbol{\mu}_k$ where $\boldsymbol{\mu}_k$ is the cluster medoid of cluster $k^*$. Rather than Euclidean distance, we use the distance matrix extracted in the previous step for assignment. The algorithm re-estimates medoid $\boldsymbol{\mu}_k$ using the points assigned to cluster $k^*$. For each point $x_i$, we check the total distance to all other points and we assign the point with minimum distance as the cluster medoid. Then, we repeat the steps until algorithm converges or reaches to a pre-specified number of runs. The time complexity of the clustering process is $O(t\ K\ N^2)$ where t is the number of iterations, K is number of clusters and N is the size of the dataset.

## 5 Experiments

### 5.1 Experimental Setup

We experimented on two synthetic datasets and seven real datasets from UCI Machine Learning Repository [1]: *Soybean, Iris, Wine, Ionosphere, Balance, Breast Cancer and Satellite.* The properties of these dataset are summarized in Table 1. $N$ is the number of instances, d is the number of dimensions, and K is the number of clusters in each dataset. We have measured the clustering accuracy as:

$$Accuracy = \sum_{i>j} \frac{1\{1\{c_i = c_j\} = 1\{\hat{c}_i = \hat{c}_j\}\}}{0.5N(N-1)} \tag{6}$$

where $1\{\cdot\}$ returns 1 if any pair of instances $x_i$ and $x_j$ are assigned correctly by the algorithm [20]. In each experimental setup, we have run the clustering algorithms for 50 times and reported the average accuracy ratios.

Must-link and cannot-link constraints are generated randomly at equal amounts and the total amount is varied proportional to the dataset size. For each run, we use the same constraint set for all algorithms. EMC parameters: we use $\alpha$=1.6 (shown through preliminary experiments to give good results), and increase the nearest neighbors and number of shortest paths proportional to the dataset size. Weights of the positive/negative constraint edge, $q_r/q_e$, are set to 1.

**Table 1.** Datasets used in experiments and running time of the algorithms (*in seconds*)

| | | *Soybean* | *Iris* | *Wine* | *Ionosphere* | *Balance* | *Breast* | *Satellite* |
|---|---|---|---|---|---|---|---|---|
| Dataset details | *N* | 47 | 150 | 178 | 351 | 625 | 683 | 4435 |
| | *d* | 35 | 4 | 13 | 34 | 4 | 9 | 36 |
| | *K* | 4 | 3 | 2 | 2 | 3 | 2 | 6 |
| Running Times (secs) | *SS-Kernel* | ~0.1 | 0.1 | 0.1 | 0.1 | 0.3 | 0.6 | 73.8 |
| | *MPCK* | 0.4 | 0.5 | 0.5 | 0.5 | 0.5 | 0.6 | 6.4 |
| | *Diagonal* | 0.2 | 0.3 | 0.6 | 1.3 | 3.6 | 5.1 | 304.3 |
| | *EMC* | 0.6 | 1.1 | 1.6 | 4.3 | 7.0 | 7.3 | 66.1 |

To visualize how our method works, we generated two synthetic datasets:

*Gaussian:* A set of 180 two-dimensional instances generated by Gaussian number generator, as shown in Figure 6. 120 instances in vertical and lower horizontal sets are labeled as class one. Upper horizontal set is label as class two.

*ThreeCircles:* Similar to *TwoCircles* data in [12], we have generated three layered circular data with 300 instances in 2 dimensions. Each circle represents one class with 100 data points in it.

We generated small amount of must-link and cannot link constraints (18 for *Gaussian* and 30 for *ThreeCircles*). Figure 6 shows final clustering of these datasets with high accuracy. For the circular data, Graph Construction process had a pre-clustering effect. As we have selected *k*-nearest neighbors for each point, there were not too many inter-cluster edges in the graph. In addition, the re-adjustment phase successfully wiped out the effects of these inter-cluster edges for the clustering phase. The rest of the experiments are run on the real datasets.

## 5.2 Effect of Parameters *k* and *p*

In this set of experiments we investigate the effect of the values of parameters *k* and *p*. *k* is the number of neighbors we pick, when converting vector data to graph data, and the number of shortest paths we use for our distance metric. For values below 5, we get heavily disconnected graphs, when converting vector data to graph data, thus experiments are not run for smaller values. We note that relatively small values of *k* (>4) are performing well enough and by using greater values we do not get significant improvements (Figure 9). On the other hand, greater values increase execution time. In summary, with small values of *k* we cannot fully take advantage of the k-shortest paths distance metric while for large values all edges are labeled as an affected edge, and consequently false escalation or reduction occurs for too many edges.

The number of partitions, *p,* has a significant effect on running time while preserving the accuracy (Figure 7). The algorithm runs up to 24x faster for the Breast dataset without significant loss of accuracy. Given that this is a small dataset, we have more gain in performance for larger datasets. After *p*=12, the accuracy starts to decline because the K-SD shortest path distance approximation does not keep up with very small-sized subgraphs. On the flip side, the running time starts increasing after p=16. The reason for this situation is the high number of hubs. The computation of matrix *S* starts to dominate the running time as it requires $O(|H|^3)$ time. One advantage of partitioning is that we no longer need to increase the value of parameter *k* and about five shortest paths are quite enough to compute distances accurately.

## 5.3 Effects of Only Must-Link or Cannot-Link Constraints

Next, we examine the effect of constraint types individually (Figure 8). When we use the must-link constraints alone, it reduces the weights of both inter-cluster and intra-cluster edges. When the reduction ratio on weights of intra-cluster edges is larger than inter-cluster edges, there is a small gain. Similarly, when cannot-link constraints are used alone, the weights of inter-cluster edges increase more than weights of intra-cluster edges. However, the gain in accuracy is greater than when using only positive edges. In that respect our algorithm's behavior differs from the one reported in [13],

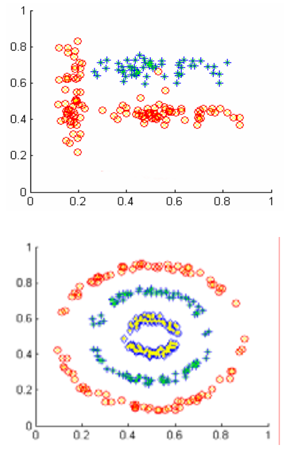


**Fig. 6.** EMC clusters (a) *Gaus-sian* and (b) *ThreeCir-cles* datasets with small set of constraints

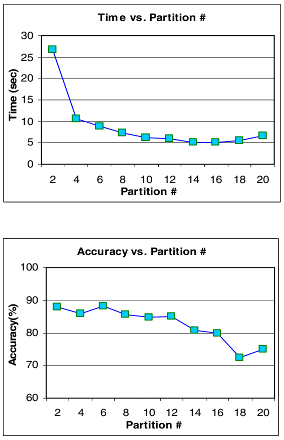


**Fig. 7.** Effect of partition # on clustering Breast dataset in terms of (a) running time and (b) accuracy

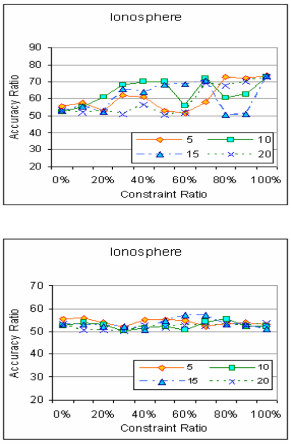


**Fig. 8.** The effect of only (a) negative edges and (b) only positive edges on Iono-sphere dataset

and this shows that the informativeness of a constraint type depends on how it is applied. Furthermore, the accuracy ratio trend is not steady as the number of constraints is augmented. On the other hand, when used together, we get optimal results for the algorithm. On incorrectly validated edges, as seen in Figure 10, they cancel the effect of each other. We observed the same phenomena for other algorithms as well.

## 5.4 Comparison with Other Techniques

We used real datasets to compare our EMC algorithm with the MPCK-Means, SS-Kernel-KMeans and KMeans+Diagonal Metric algorithms, which are publicly available online. We used the same parameters for EMC: $k$=5 and $p$=|$V$|/80 (each subgraph has approximately 80 nodes. EMC outperforms MPCK-Means, SS-Kernel-KMeans and KMeans+Diagonal Metric algorithms on all datasets, except *Breast* and *Wine (Fig. 11).* It runs better than SS-Kernel-KMeans and Kmeans+Diagonal Metric on *Breast* dataset and quite reasonable compared to the MPCK-Means. For the *Wine* dataset, graph-based methods such as SS-Kernel-KMeans and EMC do not improve the performance significantly and overall accuracy is very low compared to metric-based methods.

SS-Kernel-KMeans runs the min-cut objective while EMC tries to minimize the overall pairwise distance. The same way MPCK-Means and Kmeans+Diagonal Metric algorithms could not improve the clustering for *Balance* regardless of constraint amount, EMC fails to increase the accuracy for *Wine* dataset. In some experiments, we detect phenomena where the accuracy of the algorithm goes up and down slightly as we increase the number of constraints. As shown in [6], this is a general problem of randomly-chosen constraint sets, where some constraints reduce the clustering

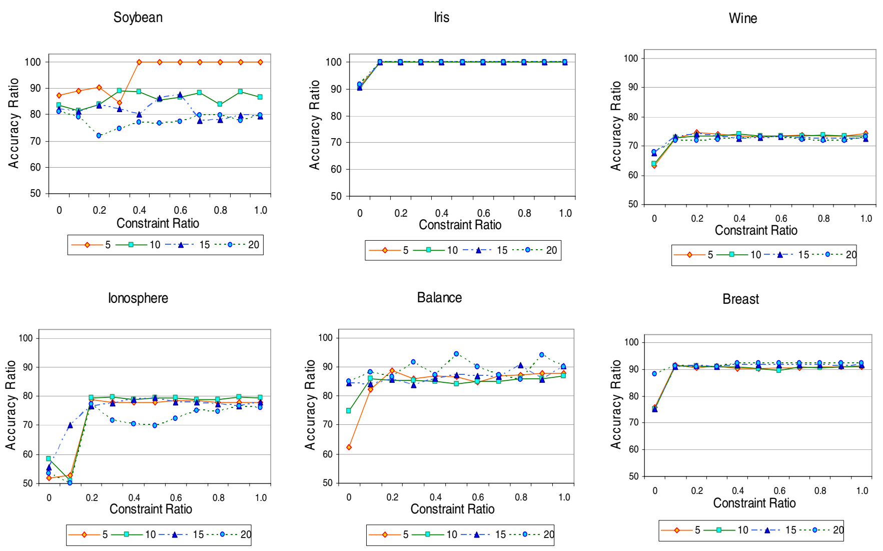


**Fig. 9.** Clustering results for EMC. The plots show the accuracy ratio achieved in respect to the constraints ratio used. The different lines stand for different values of *k* ranging from 5 to 20. Constraints amounts are *x·N* where x is the constraint ratio.

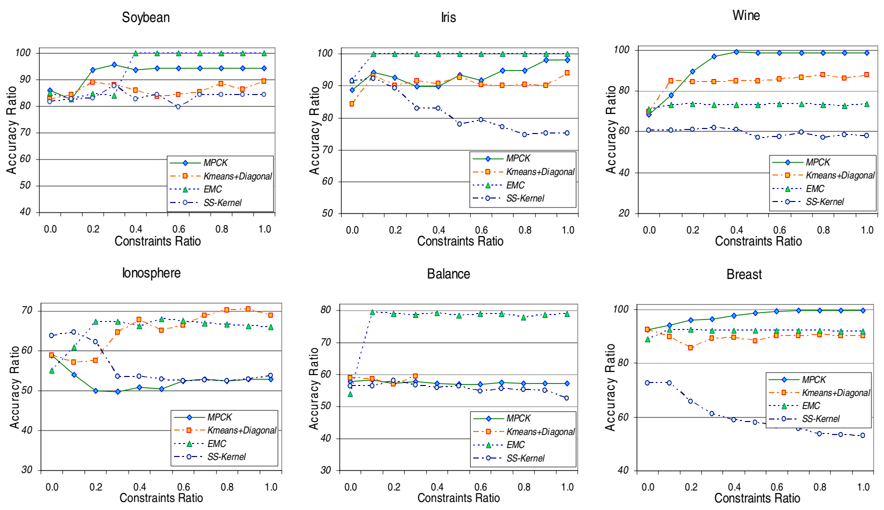


**Fig. 10.** EMC vs. MPCK-Means, KMeans+Diagonal and SS-Kernel-KMeans

performance. Thus, a learning metric or an edge weight re-adjustment method, is not always reliable for a small number of constraints. Compared to other methods, EMC is typically more trustworthy even when using few constraints.

### 5.5 Running Time Experiments

We have performed experiments on the running time of the algorithms. All experiments were carried out on 1.7 GHz Pentium IV machine with 512 MB memory. We have performed 10 experiments for each algorithm as we increase the constraint amount by 10%·*N,* where *N* is the dataset size, for each experiment and reported the average running time of these experiments. The running time of EMC is comparable to the other algorithms tested. In addition, EMC scales almost linearly with the number of partitions in practice. It is almost linear because it processes equal-sized subgraphs and number of hubs affects the linearity in time complexity during merge process. Results are given in Table 1.

## 6 Conclusions

We have presented a framework that, when given a dataset of instances and user constraints, transforms vector data into a graph and improves the clustering algorithm distance metric by adjusting the edge weights based on user constraints. The most important contribution lies in the way the weights are adjusted, based on ideas from Electromagnetic Field Theory. Instead of modifying the distance metric, it alters the distances between objects in the graph domain. EMC algorithm allows us to cluster both vector-based and graph-based datasets and it works with distances only as well. In addition to K-Medoids, we can also integrate other clustering algorithms into the framework. We have shown that even when using a small amount of constraints, the algorithm improves the clustering accuracy significantly.

**Acknowledgments.** This work was partially supported by NSF IIS-0534781, NSF 0803410, ONR N00014-07-C-0311 AWARE, Health-e-Child and SemsorGrid4Env.

## References

[1] Asuncion, A., Newman, D.J.: UCI Machine Learning Repository

[2] Bar-Hillel, A., Hertz, T., Shental, N., Weinshall, D.: Learning distance function using equivalence relations. In: ICML 2003, Washington DC (August 2003)

[3] Basu, S., Bilenko, M., Mooney, R.J.: A Probabilistic Framework for Semi-Supervised Clustering. In: KDD 2004, Seattle, WA (August 2004)

[4] Bilenko, M., Basu, S., Mooney, R.J.: Integrating Constraints and Metric Learning in Semi-Supervised Clustering. In: ICML 2004, Canada, July 2004, pp. 81–88 (2004)

[5] Brander, A., Sinclair, M.: A comparative study of k-shortest path algorithms. In: Proceedings of 11th UK Performance Engineering Workshop for Computer and Telecomm. Systems (1995)

[6] Davidson, I., Wagstaff, K., Basu, S.: Measuring Constraint-Set Utility for Partitional Clustering Algorithms. In: Proceedings of the 17th European Conference on Machine Learning, Berlin, Germany, September 18-22 (2006)
[7] Dhillon, I., Guan, Y., Kulis, B.: A fast kernel-based multilevel algorithm for graph clustering. In: Proceedings of ACM SIGKDD 2005, Chicago, Illinois, USA, August 21-24 (2005)
[8] Gallagher, B., Tong, H., Eliassi-Rad, T., Faloutsos, C.: Using ghost edges for classification in sparsely labeled networks. In: KDD 2008, Las Vegas, NV, USA, August 24-27 (2008)
[9] Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing 20(1), 359–392 (1999)
[10] Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: an Introduction to Cluster Analysis. John Wiley & Sons, New York (1990)
[11] Koren, Y., North, S.C., Volinsky, C.: Measuring and Extracting Proximity in Networks. In: KDD 2006, Philadelphia, Pennsylvania, USA, August 20-23 (2006)
[12] Kulis, B., Basu, S., Dhillon, I., Mooney, R.: Semi-supervised graph clustering: a kernel approach. In: Proceedings of the 22nd international conference on Machine learning, Bonn, Germany, August 07-11, 2005, pp. 457–464 (2005)
[13] Law, M.H.C., Topchy, A.P., Jain, A.K.: Model-based clustering with probabilistic constraints. In: SDM 2005 (2005)
[14] Lutz, H., Stocker, H., Harris, J.W.: Handbook of Physics, 1st edn., pp. 439–444 (2002)
[15] Raytchev, B., Murase, H.: Unsupervised Face Recognition from Image Sequences Based on clustering with Attraction and Repulsion. In: CVPR 2001, vol. 2, p. 25 (2001)
[16] Suhir, E.: Applied Probability for Engineers and Scientists. McGraw-Hill, New York (1997)
[17] Tong, H., Faloutsos, C., Pan, J.: Fast Random Walk with Restart and Its Applications. In: ICDM 2006, Hong Kong (2006)
[18] Wagstaff, K., Cardie, C., Rogers, S., Schroedl, S.: Constrained K-Means clustering with background knowledge. In: ICML 2001, pp. 577–584 (2001)
[19] Weber, R., Schek, H., Blott, S.: A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In: VLDB 1998, New York, USA, pp. 194–205 (1998)
[20] Xing, E., Ng, A.Y., Jordan, M., Russell, S.: Distance metric learning, with app- lication to clustering with side-information. In: Advances in NIPS, vol. 15. MIT Press, Cambridge (2002)
[21] Xu, X., Yuruk, N., Feng, Z., Schweiger, T.A.J.: SCAN: a structural clustering algorithm for networks. In: KDD 2007, San Jose, California, USA, August 12-15 (2007)
[22] Yan, B., Domeniconi, C.: An Adaptive Kernel Method for Semi-supervised Clustering. In: Proc. of the 17th European Conference on Machine Learning, Berlin, Germany (September 2006)
[23] Klein, D., Kamvar, S.D., Manning, C.D.: From Instance-level Constraints to Space-Level Constraints: Making the Most of Prior Knowledge in Data Clustering. In: Proc. of 19th Int. Conf. on Machine Learning 2002, San Francisco, CA, USA (2002)