Clustering with Relative Constraints

Eric Yi Liu
liuyi@cs.unc.edu

Zhaojun Zhang
zzj@cs.unc.edu

Wei Wang
weiwang@cs.unc.edu

Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175 USA

ABSTRACT

Recent studies [26, 22] have suggested using relative distance comparisons as constraints to represent domain knowledge. A natural extension to relative comparisons is the combination of two comparisons defined on the same set of three instances. Constraints in this form, termed *Relative Constraints*, provide a unified knowledge representation for both partitional and hierarchical clusterings. But many key properties of *relative constraints* remain unknown.

In this paper, we answer the following important questions that enable the broader application of *relative constraints* in general clustering problems:

- **Feasibility**: Does there exist a clustering that satisfies a given set of *relative constraints*? (consistency of constraints)
- **Completeness**: Given a set of consistent *relative constraints*, how can one derive a complete clustering without running into dead-ends?
- **Informativeness**: How can one extract the most informative *relative constraints* from given knowledge sources?

We show that any hierarchical domain knowledge can be easily represented by *relative constraints*. We further present a hierarchical algorithm that finds a clustering satisfying all given constraints in polynomial time. Experiments showed that our algorithm achieves significantly higher accuracy than the existing metric learning approach based on relative comparisons.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications —*Data mining*; I.5.3 [**Pattern Recognition**]: Clustering— *Algorithms*

General Terms

Algorithms,Experimentation

Keywords

Relative constraints, constrained clustering, hierarchical clustering

1. INTRODUCTION

Clustering is traditionally considered an unsupervised learning task, but side information is available in many applications which may provide extra guidance to the clustering procedure. In recent years, many papers on semi-supervised clustering or constrained clustering investigated the effect of incorporating such domain knowledge into traditional unsupervised clustering algorithms (see for survey, [8]). In these studies, domain knowledge usually appears as constraints, typically in the form of pairwise relations [30, 31]. The must-link (ML) and cannot-link (CL) constraints indicate that two data instances must or cannot belong to the same cluster. Clustering with both types of pairwise constraints has been proven to have an intractable feasibility problem: determining the existence of any partition of data instances into a given number of clusters that satisfies all the constraints is **NP**-complete [14]. Therefore, an efficient clustering algorithm that satisfies all constraints for all datasets does not exist unless **P**=**NP** [14].

Recent studies [26, 22] have suggested using relative distance comparisons to represent domain knowledge: a is closer to b than to c, or equivalently $d(a, b) < d(a, c)$ where $d(...)$ is the distance function. The rationale behind relative comparisons is as follows: a domain expert usually has some general sense of closeness (similarity) among a subset of data instances. Such knowledge of closeness may be vague and only in a relative sense, that is, we can only determine, between two instances, which one is closer to a given third instance, but we cannot determine an absolute closeness measure. Several metric learning approaches [26, 22] were developed to redefine the distance function based on input relative comparisons.

A natural extension to relative comparisons is to consider, among three data instances, which two form the closest pair. Given three instances a, b, c, we use $ab|c$ to denote that (a, b) is the closest pair. This is equivalent to two relative comparisons defined on a, b, c: $d(a, b) < d(a, c)$ and $d(a, b) < d(b, c)$. We use the term "*relative constraint*" to denote constraints in this form. *Relative constraint* is a natural extension to relative comparison and can often be used in applications of relative comparisons. This is because, when a relative comparison is used, other comparisons regarding the same three instances are often available (or attainable at low cost).

The advantage of *relative constraints* over existing constraint models is that it unifies instance-level constraints with structure-level knowledge. Traditionally, instance-level constraints fit well into partitional clustering schemes (such as *K-Means*) while their use in hierarchical clustering is unintuitive[1]. Each *relative constraint*, as a natural extension to instance-level relative comparisons, represents a local hierarchy where the closest pair are merged before the more distant instance being merged with either instance of the pair (Fig.1(b)). This local hierarchical information can be naturally exploited in a hierarchical setting. Thus, *relative constraints* provide a unified knowledge representation for both partitional and hierarchical clustering frameworks. The use of *relative constraints* also greatly expands the possible sources of domain knowledge: besides existing sources for relative comparisons, *relative constraints* can be obtained from structural knowledge such as ontology and taxonomy databases.

In [5], the authors demonstrated the potential usefulness of *relative constraints* in hierarchical text clustering (under the term *Must-Link-Before constraints*). However, unlike pairwise constraints, many important properties of *relative constraints* remain unexplored and this hinders the application of *relative constraints.* In this paper, we answer the following important questions and thus facilitate the use of *relative constraints* in general clustering problems:

- **Feasibility**: Does there exist a clustering hierarchy (and partition) that satisfies a given set of relative constraints? (consistency of constraints) Unlike pairwise constraints, we show that the feasibility problem of *relative constraints* can be solved in polynomial time. This implies the possibility of a polynomial-time clustering algorithm that satisfies all given consistent *relative constraints.*

- **Completeness**: Given a set of consistent relative constraints, how can one derive a complete clustering hierarchy (and partition) without running into dead-ends? We found that, given a set of consistent constraints, a naive hierarchical approach which does not violate any constraint in any step can still run into dead-ends. This is the scenario where the hierarchy is developed incompletely because no pair of clusters could be further merged, though other merging sequences may produce a much more complete hierarchy. This suggests that the first algorithm (iHAC) presented in [5] does not guarantee a complete hierarchy if all constraints are enforced. A similar problem has been reported for pairwise constraints [15].

- **Informativeness**: How can one extract the most informative *relative constraints* from given knowledge sources? We show that any domain knowledge in the form of tree-like hierarchies can be conveniently decomposed into *relative constraints.* When the degree of any internal node is bounded by a constant, the $O(n)$ most informative constraints can be selected, which represent the complete hierarchical knowledge of n instances. This property is particularly useful in applications such as data analysis with constraints from ontology databases (Gene Ontology [2], DBLP [23], etc.). Other examples include the topic class hierarchy in [5].

[1]In [15], the authors discussed the problems with pairwise constrains in hierarchical clustering. For relative comparisons, existing work focuses on partitional clustering.

In this paper, we propose a hierarchical algorithm named *ReCon* that finds a complete hierarchy or k-partition satisfying all given constraints whenever possible. We validated our model and algorithm with real-world datasets. Experiments with randomly generated constraints showed that our algorithm significantly outperforms the existing metric learning approach based on relative comparisons. With informative constraints that are selected systematically from existing hierarchical knowledge, the existing metric learning approach does not achieve noticeable performance gains. Our algorithm *ReCon*, on the contrary, exhibits substantial accuracy improvements.

2. RELATED WORK

2.1 Pairwise Constraints

The pairwise must-link and cannot-link constraints were first introduced in [30, 31]. Since then, many studies have been done to exploit the pairwise constraints in clustering problems. Existing methods generally fall into two categories: directly enforcing constraints in modified clustering algorithms or learning a new metric from constraints. In the first category, Wagstaff et al. proposed a variant of the widely used K-Means algorithm that can enforce constraints in cluster assignments at each iteration [31]. In [6], Basu et al. used constraints to initialize the K-Means algorithm instead of random seeding. A density-based method that incorporates pairwise constraints is given in [25]. Recently, an agglomerative hierarchical approach is proposed in [15] by Davidson and Ravi. In the second category, pairwise constraints are used in defining optimization criteria for a desired metric. Klein et al. [21] used constraints to adjust Euclidean distance by shortest-path algorithm. Xing and collaborators [32] attempted to learn Mahalanobis distance using convex optimization. The HMRF-KMeans algorithm proposed in [7] combines both constraint satisfaction and metric learning into a single probabilistic framework. MPCK-Means [9] learns an individual metric for each cluster and allows violation of constraints by imposing penalties.

2.2 Relative Distance Comparisons and Relative Constraints

Existing studies on relative comparisons focus on metric learning. In [26], an SVM-like approach is proposed to learn a weighted distance function from relative comparisons. Kumar et al. [22] proposed to learn an SVaD measure from relative comparisons. Note that existing work on relative comparisons can be used to solve clustering problems with *relative constraints* (since each relative constraint is equivalent to two relative comparisons). In [4, 5], *Relative constraints* have been investigated under the term *Must-Link-Before constraints* with a focus on text clustering. Its important properties such as feasibility and informativeness however remain largely unknown.

3. RELATIVE CONSTRAINTS

3.1 Definition of Relative Constraints

We consider, among three data instances, which two form the closest pair. Given three data instances a, b, c, we write $ab|c$ if (a, b) is the closest pair. In other words, $ab|c$ represents $d(a, b) < d(a, c)$ and $d(a, b) < d(b, c)$ where $d(...)$ is the distance function. Hence, each *relative constraint* equals two relative comparisons defined on the same set of three instances.

The *relative constraints* also unambiguously represent the knowledge of conditional cluster assignments: given $ab|c$, if a, c or b, c belong to the same cluster, then a, b, c must all belong to the same cluster. Note that the three involved instances can be arbitrarily chosen from the whole set of data instances. A constraint in the conditional form does not prevent three involved instances from being selected from the same cluster or three different clusters. In addition, the closest pair is unordered, i.e., $ab|c$ equals $ba|c$.

3.2 Relative Constraints as Rooted Triplets

The relative closeness of data instances can be represented by a rooted and unordered tree or hierarchy. In the hierarchy, each leaf represents a data instance and each internal node represents a group of instances that are similar at certain level. Instances with higher similarity are grouped at a lower level, before instances that are more distant merge into bigger groups. In this paper, we use the terms "tree" and "hierarchy" interchangeably to refer to such structure.

The local hierarchy of each *relative constraint* $ab|c$ is a rooted binary tree with three leaves as shown in Fig.1(b). In this tree, the closest pair a, b are merged first before they are merged with c. A rooted binary tree with three leaves is called a *rooted triplet*. *Rooted triplet* is an important concept first introduced in modeling phylogenies (which represent evolutionary relatedness among organisms). Representing *relative constraints* using *rooted triplets* enables us to leverage the theories and algorithms developed in phylogenetic tree studies.

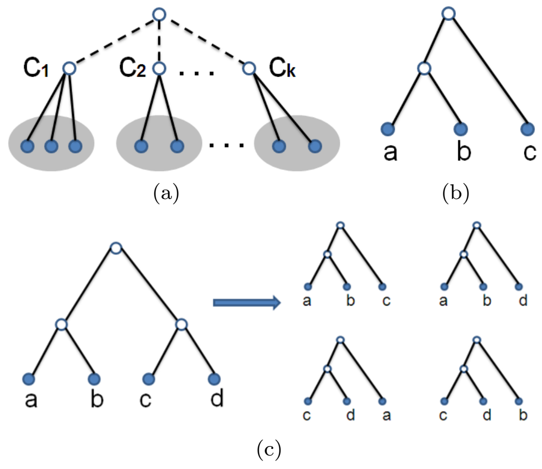

Figure 1: (a) The hierarchical representation of non-hierarchically labeled classes or partitional clustering (b) The rooted triplet representing the local hierarchy of constraint $ab|c$ (c) An exemplar hierarchy of four data instances can be decomposed into a set of rooted triplets

3.2.1 *Inducing Rooted Triplets From Bigger Hierarchy*

Here we introduce the "induce" operation for *rooted triplets*. To induce a *rooted triplet* from a bigger hierarchy, we first find the minimum subtree connecting the three involved instances. Then we contract edges whose both end-nodes are of degree two. Consider the example in Fig.1(c). We can induce the following *rooted triplets*: $ab|c$, $ab|d$, $cd|a$, $cd|b$ from the given hierarchy. If the hierarchy is not binary, it is possible to induce a subtree in which all three instances are direct children of the root. In this case, we say the three instances are unconstrained since their relative closeness is unspecified.

4. INCORPORATING RELATIVE CONSTRAINTS IN CLUSTERING

Most of existing clustering algorithms fall into two types: partitional clustering generates a k-partition of data where k is either given or automatically found; hierarchical algorithms create a global hierarchy or dendrogram from which a k-partition can be obtained. Recall that a relative constraint can be viewed as a conditional cluster assignment or a *rooted triplet*. We now define the satisfaction of relative constraints for both hierarchy and k-partition of data instances.

Definition 4.1 A hierarchy of data instances satisfies all constraints if all the corresponding *rooted triplets* can be induced from the hierarchy. If there exists at least one hierarchy that satisfies all constraints, we say that the constraints are consistent.

Definition 4.2 A k-partition satisfies all constraints if all the equivalent conditional cluster assignments are satisfied. i.e., for each constraint in the form of $ab|c$, if a, c or b, c belong to the same cluster, then a, b, c must all belong to the same cluster.

In this section, we discuss how to incorporate *relative constraints* into a hierarchical clustering framework. Our algorithm generates a hierarchy of data instances that satisfies all constraints as long as the given constraints are consistent. Given any k, a k-partition that satisfies all the constraints can then be generated by cutting the resulting hierarchy at the appropriate level.

4.1 Feasibility Under Relative Constraints

We first examine the feasibility problem for hierarchical clustering under *relative constraints*. To be more specific, given a set of data instances, does there exist a clustering hierarchy of instances that satisfies all given *relative constraints*?

4.1.1 *Existence of Hierarchy*

Since *relative constraints* can be represented by *rooted triplets*, the above problem is equivalent to the problem of determining the existence of a supertree: given a set of *rooted triplets*, is there a tree from which all *rooted triplets* can be induced? We denote the set of data instances by D. The set of constraints is denoted by C. The set of instances involved in constraints is denoted by D_C. Obviously, we only need to determine whether there is a supertree T for D_C that satisfies all *relative constraints* in C. Data instances not involved in any constraints can be arbitrarily added as leaves in T without violating any constraints.

The existence problem of supertree can be solved by the *OneTree* algorithm initially given in [1]. Several variants of *OneTree* algorithm have been proposed [24, 10]. Below is a simplified version of the *OneTree* algorithm presented in [24]. The algorithm operates by transforming all constraints into a graph representation: a vertex is created for each instance; an edge connects two vertices a, b if there is any constraint of the form $ab|*$. It then tests the connectivity of the graph: if the graph is disconnected, then, for each connected component, it finds the subset of instances in the component and the subset of constraints defined only on these instances. These subsets of instances and constraints are then tested by a recursive call of the algorithm. This recursive procedure is based on the property that a set of constraints is consistent if and only if any sub-graph defined on more than two instances is disconnected. The algorithm returns a hierarchy that satisfies all constraints if any such hierarchy exists. The complexity of the algorithm is $O(|D_C| \times |C|)$. The algorithm does not exhaust all possible hierarchies that satisfy C.

```
1. Function OneTree($C$,$D$)
2. Input: set $C = \{r_1, ..., r_m\}$ of relative constraints,
3.        set $D = \{d_1, ..., d_n\}$ of data instances
4. Output: a hierarchy $T$ of $D$ that satisfies $C$
5. If $|D| = 1$, return a single node labelled by $d_1$
6. If $|D| = 2$, return a tree with two leaves $d_1$ and $d_2$
7. Create sets $D_i = \{d_i\}$, $i = 1..n$
8. For each constraint $(ab|c) \in C$, merge the sets $D_i, D_j$
   containing $a, b$.
9. If there is only one set left, return fail
10. For each remaining set $D_i$ do
11.     Let $C_i$ be the set of constraints of which all three
        instances are in $D_i$
12.     If OneTree($C_i$,$D_i$) returns a tree then call it $T_i$
        else return fail
13. Construct a tree $T$ by connecting the roots of all $T_i$ to
    a new root
14. Return $T$
```

4.1.2 *k-partition by Cutting Hierarchy*

Given any clustering hierarchy satisfying all constraints, we can always cut the hierarchy to form a k-partition. The cutting starts from the root level and goes down following the recursion levels in *OneTree* algorithm. The resulting k-partition always satisfies all constraints. Due to its simplicity, we omit the proof here.

4.2 A Clustering Algorithm That Generates A Complete Hierarchy

The above *OneTree* algorithm only ensures the satisfaction of constraints. It does not consider instance attributes or incorporate any metric function in generating hierarchy. The hierarchy returned is determined only by the graph representation of constraints. Hence, the *OneTree* algorithm cannot be used as a hierarchical clustering algorithm.

We now present our modified hierarchical agglomerative algorithm that incorporates *relative constraints* in the clustering procedure. Given a set of n instances, the traditional agglomerative algorithm starts with n clusters where each cluster contains only one instance. At each round, two closest clusters are merged. The algorithm completes when there is only one cluster left. Now with *relative constraints* incorporated, we want to ensure that: (1) we do not violate any constraint (2) we generate a complete hierarchy as long as the constraints are consistent.

The first condition can be easily satisfied: at each round, for any constraints in the form of $ab|c$, if a, b do not belong to the same cluster, we do not merge the clusters containing b, c or a, c. The satisfaction of the second condition is however obscure: if we do not violate any constraint in each round, do we always get a complete hierarchy? The answer is unfortunately no. It is possible that, though given constraints are not violated at any previous round, the hierarchical clustering algorithm runs into a dead-end where any further merging violates some given constraint. A simple example consists of four data instances a, b, c, d and two constraints $ab|c$, $cd|a$. There exists a complete hierarchy as in Fig.1(c). If b, d are merged according to some metric in the first round, there is no possible merging move in the second round.

In the above example, the constraint $ab|d$ can actually be inferred from the two given constraints. With $ab|d$ inferred, merging b, d in the first round becomes an invalid choice. One may think of pre-computing all inferable constraints to determine the eligible merging choices at each round. The set of all constraints that can be inferred is called the *closure* of the given constraints. Unlike pairwise constraints, the closure of *relative constraints* cannot be trivially computed. To be more specific, we cannot repeatedly apply low-order inference rules to derive the complete closure. This is due to the existence of irreducible inference rules for *rooted triplets* [10]: for any $n > 3$, there are inference rules of order n that cannot be derived from rules involving fewer than n *rooted triplets*. The property prevents us from quickly pre-computing all eligible merging steps at each round.

To avoid running into dead-ends, we resort back to the *OneTree* algorithm which tests the existence of any complete hierarchy. At each round, before merging two clusters, we test whether there still exists a solution if the two are merged. A merging step is compatible with constraints only if it does not change the existence of a complete solution. Our complete algorithm *ReCon* is presented below.

```
1. Algorithm ReCon
2. Input: set $C = \{r_1, ..., r_m\}$ of relative constraints,
3.        set $D = \{d_1, ..., d_n\}$ of data instances
4. If OneTree($C$,$D_C$) returns fail then return fail
5. Create clusters $CL_i = \{d_i\}$, $i = 1..n$
6. While number_of_clusters $> 1$ do
7.     Let $S = \{i : CL_i \ remains\}$
8.     Find the closest pair of clusters $CL_i, CL_j$ s.t.
9.         Neither $(i * |j)$ or $(j * |i)$ is in $C$
10.          and
11.        TestCompatibility($C, S, i, j$) is true
12.    Merge $CL_i, CL_j$ into $CL_i$
13.    UpdateConstraints( $i, j$ )
```

```
1. Function UpdateConstraints($i$,$j$)
2. Input: $i, j$ are the two clusters merged
3. For each constraint $(ab|c) \in C$ do
4.     If any of $a, b, c$ is equal to $j$ then update it to be $i$
5.     If $a = b$ then remove this constraint
```

```
1. Function TestCompatibility($C'$,$S$,$i$,$j$)
2. Input: set $C'$ of relative constraints,
3.        set $S$ of cluster indices
4.        $i, j$ are the two clusters to merge
5. Output: true if merging $i, j$ is compatible with $C'$
6. If $|S| = 2$, return true
7. Create sets $S_e = \{e\}$ for each $e \in S$
8. Merge $S_i$ and $S_j$
9. For each constraint $(ab|c) \in C'$ do
10.     Merge the sets $S_e, S_f$ containing $a, b$.
11. If there is only one set left then return false
12. Let $S_e$ be the set containing $i, j$
```

13. Let C_e be the set of constraints of which all three instances are in S_e
14. Return *TestCompatibility*(C_e, S_e, i, j)

The *TestCompatibility* function is modified from the *OneTree* algorithm to allow more efficient tests. Note that the worst-case complexity of our *TestCompatibility* function is the same as that of *OneTree* ($O(|D_C| \times |C|)$). But generally *TestCompatibility* can finish much faster (at each level, it only examines one connected component of the graph representation of constraints). The actual complexity of our algorithm depends on the specific choice of merging strategy and other implementation choices. In this paper, we adopt the centroid-based hierarchical clustering based on a priority queue implementation. The overall worst-case complexity of our algorithm is $O(|D|^2 log|D| + |D| \times |D_C|^3 \times |C|)$. This is based on the extreme assumption that any pair of clusters that are both involved in constraints always have shorter distance measures (than that of any other pairs). In real applications, this worst-case scenario is very unlikely to happen and our algorithm can efficiently handle large datasets with thousands of constraints(Sec.5.3). If more efficient component tracking techniques are adopted [17, 18], the worst-case complexities of *TestCompatibility* and the whole algorithm can be further reduced to $O(|D_C| log^2|C|)$ [19] and $O(|D|^2 log|D| + |D| \times |D_C|^3 log^2|C|)$, respectively.

4.2.1 Handling Outliers in Getting k-partitions

Although the resulting binary-hierarchy embeds all given constraints, it does not guarantees a perfect k-cutting in the presence of outliers. The outliers do not merge with their expected clusters before real clusters get merged. This can potentially be solved by using more appropriate metric functions and placing constraints on the branch size. In this work, we adopt a simple branch-size threshold in cutting the hierarchy: branches with size smaller than a given threshold are considered outliers and are first ignored in the k-cutting process. After k clusters are obtained, outlier branchs are merged with their nearest clusters.

4.3 Informative Constraints from Hierarchical Knowledge

As discussed in previous sections, the use of *relative constraints* greatly expands the possible sources of knowledge. Besides existing instance-level sources for relative comparisons (such as manually labeled classes and relations), *relative constraints* as local hierarchies can be extracted from structural knowledge. Examples of such structural knowledge include ontology/taxonomy databases such as Gene Ontology [2] and DBLP [23]. Other examples include the topic class hierarchy in [5]. These knowledge resources are often in the form of tree-like hierarchies defined on a partially overlapping set of data instances. (Non-hierarchically labeled classes can also be represented by a two-level hierarchy by adding a dummy root (Fig.1(a)).) In this section, we discuss how to extract the most informative constraints from a given hierarchical knowledge.

Traditionally, constraints are generated at the instance level. Given a hierarchy of instances or a set of labeled instances, one can enumerate all possible triple-wise combinations and induce the corresponding relative constraints. This straightforward approach leads to an extraordinarily large constraint space. If the given hierarchical knowledge involves n data instances of interests, one would get $O(n^3)$ constraints. A randomly sampled subset of constraints could contain limited information due to redundancy in information. In [5], the authors noted this problem and tried to address this by first grouping similar instances before generating constraints. However, a significant amount of information can be lost in this preprocessing step. Also, parameters and metrics need to be defined for specific tasks.

Since each *relative constraint* can be represented by a *rooted triplet*, we can leverage the existing theories and algorithms in phylogenetic tree studies. It is possible to use a much smaller set of informative constraints to represent the whole set of constraints (equivalently, the whole set can be inferred from the smaller subset). For binary hierarchies, it can be shown that $n - 2$ constraints are enough to capture all information. A linear-time algorithm for finding the minimum set of *rooted triplets* from a binary hierarchy is given in [13]. The most informative *relative constraints* can thus be generated accordingly. For general hierarchies, the polynomial-time algorithm given in [24] can be easily modified to decompose a general hierarchy into $O(n)$ constraints if the degree of all internal nodes are bounded. Due to space limitations, we omit the details of the decomposition algorithm here.

5. EXPERIMENTS

In this section, we show the effect of our model and algorithm on datasets of various sizes from the UCI repository [3]. Note that our algorithm does not assume any specific metric function or merging strategy. For the evaluations presented, we explored both Euclidean distance and a learned metric based on [26]. In our implementation, we used the centroid-based hierarchical clustering. Any other merging strategies and distance measures can be incorporated as well (e.g., the inverse of sample covariance when the data is assumed to be Gaussian). The clustering accuracies in experiments are reported in the pairwise F-measure:

$$Precision = \frac{\#PairsCorrectlyPredictedInSameCluster}{\#TotalPairsPredictedInSameCluster}$$

$$Recall = \frac{\#PairsCorrectlyPredictedInSameCluster}{\#TotalPairsInSameCluster}$$

$$F\text{-}Measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

We give results for six real-world datasets from the UCI repository [3]. Table 1 lists the number of instances, dimensions and clusters of these datasets. The *Digits-389* and *Letters-IJLT* datasets are subsets from the original *Digits* and *Letters* handwritten character recognition datasets. The two subsets contain the character classes that are considered difficult to distinguish. We evaluated our algorithm with both randomly selected constraints and informative constraints derived from a knowledge hierarchy.

Dataset	#instances	#dimensions	#clusters
Iris	150	4	3
Wine	178	13	3
Ionosphere	351	34	2
Transfusion	748	5	2
Digits-389	3165	16	3
Letters-IJLT	3059	16	4

Table 1: The datasets used in the experiments

We compared our method with an existing metric learning method [26][2]. Each relative constraint can be viewed as two relative distance comparisons. A new metric function was learned from these comparisons by optimizing an SVM-like quadratic function. We set the regularization parameter C to be 1 and $A = I$ following the examples in [26]. We then solved the optimization problem using SVM-light [20]. We refer to the learned metric as "*SVM-Metric*". For each dataset, we compared the following clustering schemes:

- *ReCon + Euclidean Metric*
- *ReCon + SVM-Metric*
- *K-Means + SVM-Metric* (use learned metric alone)

We also included two unsupervised clustering schemes as baselines:

- *Hierarchical + Euclidean Metric*
- *K-Means + Euclidean Metric*

The first unsupervised approach serves as the baseline for *ReCon*. The second provides baseline for *K-Means + SVM-Metric*.

5.1 Randomly Sampled Constraints

In our experiments with randomly sampled constraints, the number of constraints ranges from 0 to $|D|$ where $|D|$ is the number of data instances. For these non-hierarchical datasets with known cluster labels, we do not know the true metric that separates the clusters (or if such metric exists). Thus, we randomly sampled constraints in the form of $ab|c$ where a, b are from the same cluster and c is from a different cluster. Our algorithm does not take this as must-link or cannot-link information and only consider this as a, b being closer than b, c and a, c. For each setting, we took the average of 50 runs of different random constraints. Fig.2 shows the accuracies measured with the five clustering schemes.

In most experiments, incorporating *SVM-Metric* in *K-Means* performed significantly better than its baseline (*K-Means + Euclidean Metric*). This proves that the learned *SVM-Metric* is very useful in representing domain knowledge. We notice that the accuracy curves of *K-Means + SVM-Metric* are relatively flat. This indicates that using *SVM-Metric* alone can get "saturated" soon and the performance does not improve with increasing numbers of constraints fed. We conjecture that this is due to its SVM-like optimization: the few support vectors representing the metric remained mostly unchanged unless there was a big change in the distribution of training data.

Our algorithm *ReCon*, incorporating either of the metrics, outperformed *K-Means + SVM-Metric* by large margins in all but one experiments. The only exception is *Iris*. This is probably because, being a relatively clean dataset, *Iris* can be well described by a metric defined on few dimensions. The advantage of our method on the remaining five datasets was generally obvious with a small amount of constraints regardless of the baseline accuracies (without constraints). It is interesting that *ReCon + Euclidean Metric* generally performed better than *ReCon + SVM-Metric*. We speculate that this is because the learned *SVM-metric* was optimized from a boundary supported by major data distribution but violating some of the constraints fed. Thus, when *ReCon* tried to enforce all given constraints, it led to conflicts and lowered the overall clustering performance.

We also notice that, despite the significant average improvement, the clustering quality in individual runs could vary considerably when applying different sets of randomly sampled relative constraints. An example is shown in Fig.3 where the min, max and average accuracies of *ReCon + Euclidean Metric* on *Digits-389* and *Letters-IJLT* are included. Similar phenomena have been observed in applying pairwise constraints using various existing approaches [16]. Note that the variance of using *SVM-Metric* alone is lower due to its "saturation" behavior. But this is at the cost of having much lower overall accuracy.

In real-world problems, it is thus important to quantify the potential benefit of given relative constraints. Davidson et al. proposed two measures, *informativeness and coherence*, to capture the potential benefit brought by pairwise constraints to a given algorithm/metric [29]. The measures are intuitive and easy to compute. But they are still insufficient in explaining the complicated behavior of constrained clustering. For example, the information redundancy of different constraints is not modeled and the coherence model in metric space is oversimplified. The situation gets more complicated with *relative constraints* where triple-wise relations are used. In the next section, we show that the benefit of constraints can be maximized when informative constraints are extracted from existing hierarchical knowledge.

5.2 Informative Constraints from a Given Hierarchy

The above experiments are based on randomly generated constraints of which the informativeness is difficult to quantify. But, as discussed in Sec.4.3, existing domain knowledge in the form of hierarchy can be effectively decomposed into a small set of constraints. Thus, given a hierarchy involving n instances, we can always find the most informative constraints that capture all topological information in the hierarchy.

To demonstrate the effectiveness of this property, we consider the two-level hierarchy formed using true cluster labels (Fig.1(a)). We used a simple method to generate $(k-1) \times |D|$ constraints that represents the two-level hierarchy. The number of clusters, k, in our experiment is between 2 and 4. Other general hierarchies can be decomposed based on the method in [24] as discussed in Sec.4.3.

```
1. Function GenerateConstraints($C$,$D$)
2. Input: a set of clusters $CL_1$ ... $CL_k$
3.         each cluster $CL_i = \{d^i_1, ..., d^i_n\}$ where $d^i_j$ is a
   data instance in this cluster
4. Output: set $C$ of relative constraints
5. For each cluster $CL_i$
6.     For each data instance $d^i_j$ in $CL_i$
7.         For each cluster $CL_l$ other than $CL_i$
8.             Add $(d^i_1 d^i_j | d^l_1)$ into $C$
9. Return $C$
```

Fig.4 shows the results of applying these informative constraints. As in previous experiments, each accuracy curve is an average of multiple runs. When applying random subsets of the selected constraints, we observed stable and consistent improvements with *ReCon*. Similar to our previous experiments, *ReCon* showed significant improvements over

[2]We did not include [5] in comparison due to its focus on text clustering.

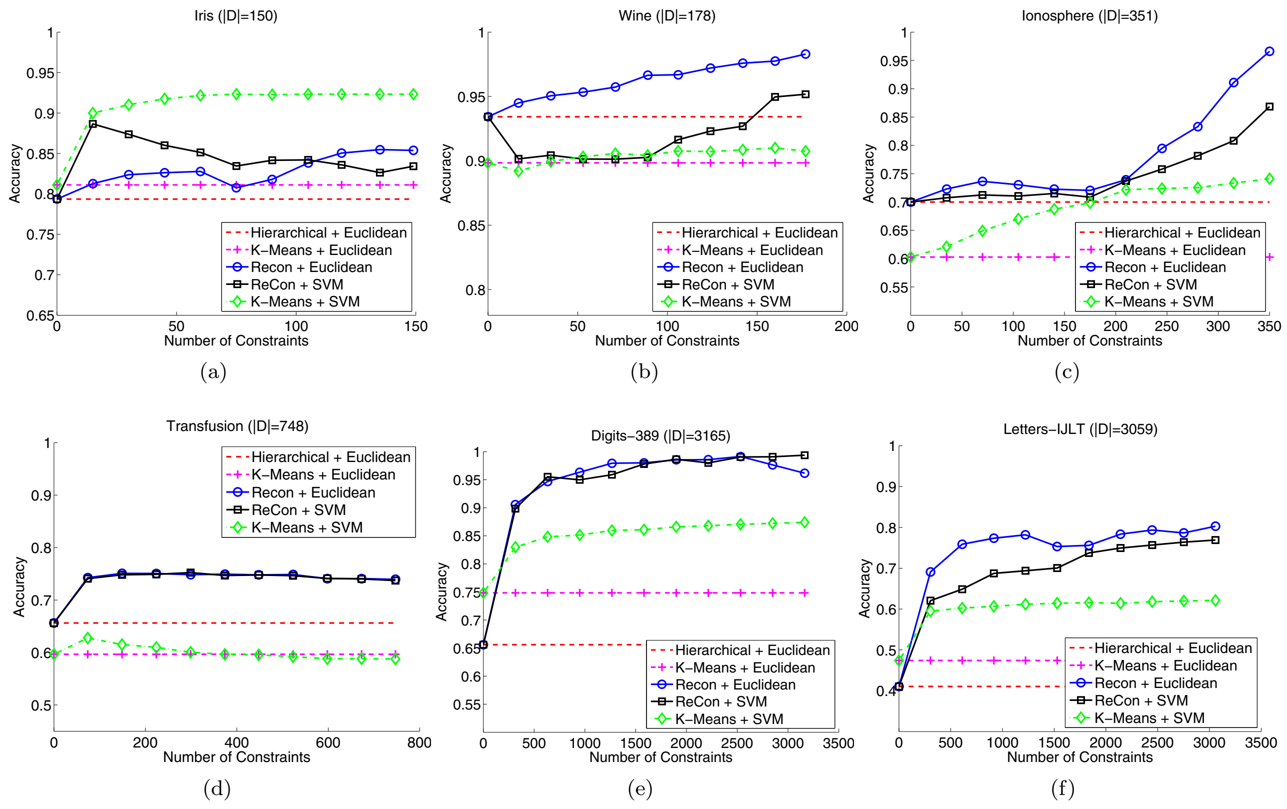

Figure 2: The clustering accuracy for 6 UCI datasets with randomly generated relative constraints. For each dataset, the number of constraints ranges from 0 to the number of instances in the dataset.

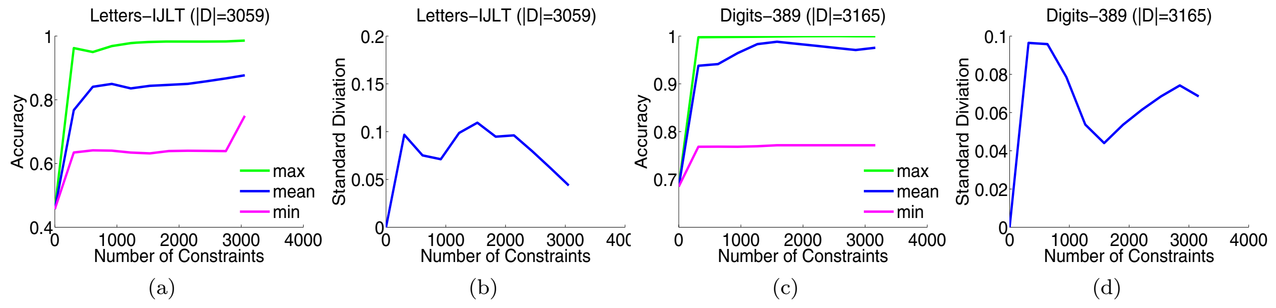

Figure 3: The variance of accuracy of 50 different runs of *Letters-IJLT* and *Digits-389*. Randomly generated constraints are applied with *ReCon + Euclidean Metric* in each run: (a) Min, max and average accuracy of *Letters-IJLT* (b) Standard deviation of *Letters-IJLT* (c) Min, max and average accuracy of *Digits-389* (d) Standard deviation of *Digits-389*

using *SVM-Metric* alone in all experiments (including *Iris* this time). With all the constraints incorporated, *ReCon* always created a binary-hierarchy that embeds the original two-level hierarchy. In the resulting k-partition, we achieved 100% accuracy on all six datasets. In comparing with using randomly generated constraints of the same amount, *ReCon* attained significantly higher accuracy with informative constraints. The variance of the performance gain was also much smaller which indicates the steady informativeness of these constraints. The effect of informative constraints is however uncertain with the metric learning approach: *K-Means + SVM-Metric* tended to reach its "saturation" with fewer constraints in *Iris* and *Digits-389*. It also achieved higher accuracy in *Transfusion*. But the overall accuracy gain in all experiments was very limited and the clustering results became worse in *Ionosphere*.

In real-world problems, the true class labels or cluster assignments are apparently unavailable. But, if there exists reliable hierarchical knowledge of a subset of instances, we demonstrate a promising way to select the most informative relative constraints. Such informative set of constraints can be fully utilized in our algorithm while the metric learning approach does not benefit from this informativeness.

5.3 Running Time

Fig.5(a) and Fig.5(b) show the running time of *ReCon + Euclidean Metric* on *Letters-IJLT* and *Digits-389*. Both datasets have more than 3,000 instances and the number

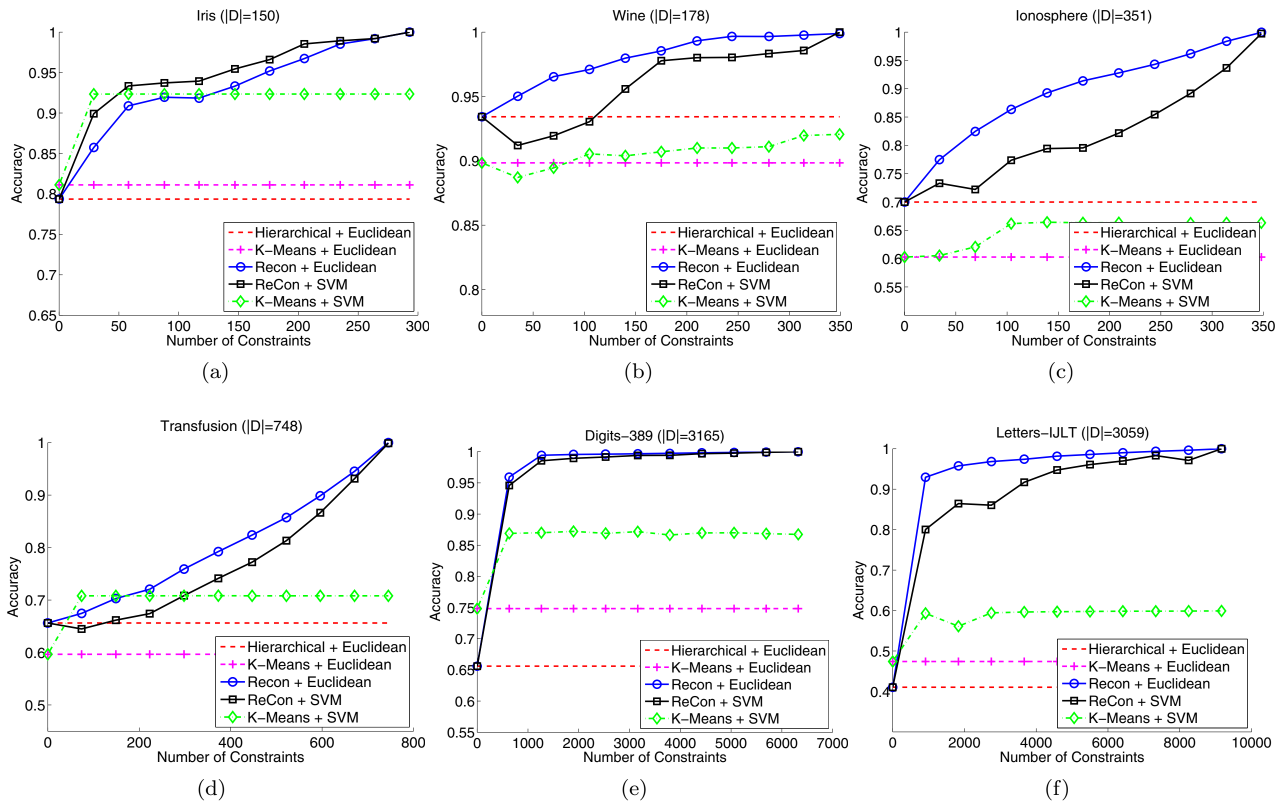

Figure 4: The clustering accuracy for 6 UCI datasets with informative constraints. For each dataset, $(k-1) \times |D|$ constraints are selected where k is the number of clusters and $|D|$ is the number of instances in the dataset. We evaluate the clustering accuracy with random subsets and complete set of the selected constraints.

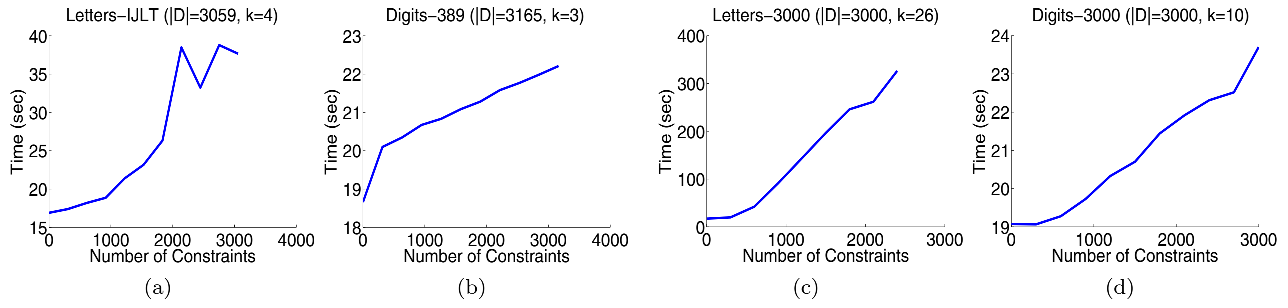

Figure 5: The running time of *ReCon + Euclidean Metric* on *Letters-IJLT*, *Digits-389*, *Letters-3000* and *Digits-3000*.

of constraints applied ranges from 0 to 3,000. Experiments were conducted on a PC with 2.6GHz Intel CPU and 8G RAM. The time plotted is the average of 50 runs of independently sampled constraint sets. The running time largely depends on the distribution of the data and the constraints selected. For comparison, we include two plots of datasets of large number of clusters (Fig.5(c) and 5(d)). *Letters-3000* has 3,000 instances randomly sampled from all 26 classes in the original Letters dataset. *Digits-3000* is sampled from all 10 classes in Digits dataset. In most experiments we conducted with the UCI datasets, our algorithm completed clustering within a few minutes.

6. CONCLUSIONS

In this paper, we investigated the important properties of *relative constraints*, an extension to relative comparisons to represent domain knowledge. Each *relative constraint* describes the local hierarchy of three instances. Besides existing instance-level knowledge sources, informative constraints can be systematically obtained from hierarchical knowledge resources such as Gene Ontology [2] and DBLP [23]. This property is desirable in making constrained clustering practical. We also present an efficient hierarchical algorithm that finds a complete hierarchy (and its corresponding partition) satisfying all given *relative constraints*. Experiments on real-world data showed promising results of applying *relative constraints*. Our algorithm significantly outperforms the metric learning approach based on relative comparisons. In addition, our algorithm exhibits unique advantage in utilizing informative constraints that are systematically extracted.

In the future work we plan to investigate how to integrate *relative constraints* with other existing constraint models. For example, pairwise constraints can be present together with *relative constraints* in many problems.

Handling Noisy Constraints

In this paper, we assume a reliable source of consistent constraints and focus on the hard-satisfaction of all given constraints. In reality, domain knowledge may contain errors and the obtained constraints can be noisy (or even become inconsistent). Hence, handling noisy/inconsistent constraints is an important step in applying *relative constraints*. If all constraints are of equal weight, one obvious approach to address inconsistency is to derive a maximum subset of consistent constraints from the given inconsistent constraints. This reduces to the Maximum Triplet Consistency problem [11] and its approximations [27, 28, 12]. Also, metric learning approaches can be used as a subroutine to filter noisy constraints. We plan to explore a more effective way to combine our framework with existing metric learning studies.

Acknowledgments

The authors would like to thank the reviewers for their valuable comments. This work was partially supported by NSF grants IIS-0812464 and IIS-0448392.

7. REFERENCES

[1] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.

[2] M. Ashburner and et. al. Gene ontology: tool for the unification of biology. the gene ontology consortium. *Nature genetics*, 25(1):25–29, May 2000.

[3] A. Asuncion and D. Newman. UCI machine learning repository, 2007.

[4] K. Bade and A. Nurnberger. Personalized hierarchical clustering. In *WI '06*, pages 181–187, 2006.

[5] K. Bade and A. Nurnberger. Creating a cluster hierarchy under constraints of a partially known hierarchy abstract. In *SDM '08*, pages 13–24, 2008.

[6] S. Basu, A. Banerjee, and R. Mooney. Semi-supervised clustering by seeding. In *ICML '02*, pages 19–26, 2002.

[7] S. Basu, M. Bilenko, and R. J. Mooney. A probabilistic framework for semi-supervised clustering. In *KDD '04*, pages 59–68, 2004.

[8] S. Basu, I. Davidson, and K. L. Wagstaff. *Constrained clustering: advances in algorithms, theory, and applications*. Chapman and Hall/CRC, 2008.

[9] M. Bilenko, S. Basu, and R. J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *ICML '04*, page 11, 2004.

[10] D. Bryant and M. Steel. Extension operations on sets of leaf-labelled trees. *Advances in Applied Mathematics*, 16(4):425–453, 1995.

[11] J. Byrka, S. Guillemot, and J. Jansson. New results on optimizing rooted triplets consistency. In *ISAAC '08*, 2008.

[12] D. Chen, O. Eulenstein, D. Fernandez-Baca, and M. Sanderson. Minimum-flip supertrees: complexity and algorithms. *IEEE/ACM transactions on computational biology and bioinformatics*, 3(2):165–73, 2006.

[13] M. Constantinescu and D. Sankoff. An efficient algorithm for supertrees. *Journal of Classification*, 12(1):101–112, 1995.

[14] I. Davidson and S. Ravi. Clustering with constraints: Feasibility issues and the k-means algorithm. *In SDM '05*, page 138, 2005.

[15] I. Davidson and S. Ravi. Using instance-level constraints in agglomerative hierarchical clustering: theoretical and empirical results. *Data Mining and Knowledge Discovery*, 18(2):257–282, 2008.

[16] I. Davidson, K. Wagstaff, and S. Basu. Measuring constraint-set utility for partitional clustering algorithms. *LNAI*, 4213:115–126, 2006.

[17] M. R. Henzinger, V. King, and T. Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. *Algorithmica*, 24(1):1–13, 1999.

[18] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM*, 48(4):723–760, 2001.

[19] J. Jansson, J. H. Ng, K. Sadakane, and W. K. Sung. Rooted maximum agreement supertrees. *Algorithmica*, 43(4):293–307, 2005.

[20] T. Joachims. Making large-scale support vector machine learning practical. *Advances in kernel methods: support vector learning*, pages 169–184, 1999.

[21] D. Klein, S. Kamvar, and C. Manning. From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In *ICML '02*, pages 307–314, 2002.

[22] N. Kumar and K. Kummamuru. Semisupervised clustering with metric learning using relative comparisons. *IEEE Transactions on Knowledge and Data Engineering*, 20(4):496–503, 2008.

[23] M. Ley. The DBLP computer science bibliography: Evolution, research issues, perspectives. In *SPIRE '02*, pages 1–10, 2002.

[24] P. Meei and N. Wormald. Reconstruction of rooted trees from subtrees. *Discrete Applied Mathematics*, 69(1):19–31, 1996.

[25] C. Ruiz, M. Spiliopoulou, and E. Menasalvas. C-dbscan: Density-based clustering with constraints. In *RSFDGrC '07*, pages 216–223, 2007.

[26] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In *NIPS '04*, 2004.

[27] C. Semple and M. Steel. A supertree method for rooted trees. *Discrete Applied Mathematics*, 105(1-31-3):147–158, 2000.

[28] S. Snir and S. Rao. Using max cut to enhance rooted trees consistency. *IEEE/ACM transactions on computational biology and bioinformatics*, 3(4):323–33, 2006.

[29] K. Wagstaff, S. Basu, and I. Davidson. When is constrained clustering beneficial, and why? In *AAAI '06*, pages 59–60, 2006.

[30] K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *ICML '00*, pages 1103–1110, 2000.

[31] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrödl. Constrained k-means clustering with background knowledge. In *ICML '01*, pages 577–584, 2001.

[32] E. Xing, A. Ng, M. Jordan, and S. Russell. Distance metric learning with application to clustering with side-information. In *NIPS '02*, pages 505–512, 2002.