Incorporating SAT Solvers into Hierarchical Clustering Algorithms - An Efficient and Flexible Approach

Sean Gilpin
Department of Computer Science
University of California, Davis
Davis, CA 95616
sagilpin@ucdavis.edu

Ian Davidson
Department of Computer Science
University of California, Davis
Davis, CA 95616
indavidson@ucdavis.edu

ABSTRACT

The area of constrained clustering has been actively pursued for the last decade. A more recent extension that will be the focus of this paper is constrained hierarchical clustering which allows building user-constrained dendrograms/trees. Like all forms of constrained clustering, previous work on hierarchical constrained clustering uses simple constraints that are typically implemented in a procedural language. However, there exists mature results and packages in the fields of constraint satisfaction languages and solvers that the constrained clustering field has yet to explore. This work marks the first steps towards introducing constraints satisfaction languages/solvers into hierarchical constrained clustering. We make several significant contributions. We show how many existing and new constraints for hierarchical clustering, can be modeled as a Horn-SAT problem that is easily solvable in polynomial time and which allows their implementation in any number of declarative languages or efficient solvers. We implement our own solver for efficiency reasons. We then show how to formulate constrained hierarchical clustering in a flexible manner so that any number of algorithms, whose output is a dendrogram, can make use of the constraints.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data Mining

General Terms

Algorithms, Experimentation

1. INTRODUCTION AND MOTIVATION

The area of clustering with constraints was introduced to the machine learning and data mining fields by Wagstaff and Cardie [18] a decade ago. During the past ten years much research has occurred in the field with the seminal paper [18] obtaining over five hundred citations and a recent collected edition [4] describes many innovative applications. Despite the great attention given to constrained clustering two noticeable "gaps" in the research have occurred. Firstly, despite an extensive survey [13] of application journals in domains such as biology, chemistry and physics showing that **hierarchical, not non-hierarchical, clustering** is the most prevalent clustering approach, little effort has looked at constrained hierarchical clustering. Secondly, the connection with the closely related computer science fields of constraint programming languages, declarative languages, and constraint satisfaction solvers has received little attention.

The first gap/limitation described above greatly limits the use of constrained clustering algorithms since in that survey it was found that over 50% of applications of clustering were to create hierarchies. This is not too surprising given that hierarchies are of more use to practitioners since they offer a more comprehensive summary of the data and can model temporal relationships such as evolution. The introduction of constrained hierarchies offers even more benefits since hierarchical techniques tend to be used in areas where considerable domain expertise already exists and hence should provide a ready source of constraints if they are rich enough to model the concepts in these fields.

The second gap/limitation is perhaps the most startling given the aim of constrained clustering is to find clusterings consistent with a given set of constraints. Modeling constraints in a procedural language is greatly limiting and gave rise to declarative languages with built-in constraint satisfaction techniques such as resolution thereby freeing the programmer to model the constraints rather than designing algorithms to solve them. A severe limitation to introducing more complex constraints beyond the seminal constraints `must-link` and `cannot-link` is that determining if they can be efficiently satisfied, and how to satisfy them becomes a difficult issue. **Our first order logic formulation, can be propositionalized into a set of Horn-SAT constraints allowing us to check the consistency of the constraints and find dendrograms that satisfy all of the supplied constraints in polynomial time.**

We make the first steps to combining hierarchical constrained clustering and constraint programming languages and solvers. Our previous work [8] explored using constraint satisfaction techniques and 2-SAT for non-hierarchical clustering in a limited setting ($k = 2$). The direction we take in this paper has many immediate and long term benefits to the field, some of which we now list.

- The domain-expert/user can specify a collection of **different types of constraints**. Previous work has focused on only one type of instance-level constraint.
- The constraints are capable of expressing hierarchical information including partial dendrograms.
- The domain-expert/user can specify a large collection of constraints and they can be efficiently checked for inconsistencies and satisfaction unlike the non-hierarchical case which is in general NP-Complete [7].

KDD'11, August 21–24, 2011, San Diego, California, USA.

- Extensive complexity and algorithm design results have created a rich literature of which normal forms of clauses are efficiently solvable and algorithms exist to solve large collections of clauses. For example, in this paper we make use of the fact that clauses in Horn-form are efficiently solvable and base our own solver on the work of Dowling and Gallier [9].
- The user and algorithm designer are free to focus on data mining algorithms and not how to model or satisfy constraints.
- By separating out the constraint satisfaction part of the algorithm and formally modeling it, we create a reusable framework that can be used with a variety of algorithms that build hierarchies.

We begin this paper by describing previous work, which is limited to just six papers, in Section 2. In Section 3 we describe our first order logic language to model constraints. Section 4 shows how an agglomerative clustering algorithm can use our formalism to find solutions to constrained hierarchical clustering problems. The description of our experiments and results can be found in Section 5.

2. PREVIOUS WORK

Despite the popularity of applying hierarchical clustering algorithms, little work has focused on hierarchical constrained clustering. Previous work has explored using instance level constraints for hierarchical clustering. Davidson and Ravi use `must-link` (ML) and `cannot-link` (CL) constraints previously used for non-hierarchical clustering and demonstrated their use for hierarchical clustering [6, 7]. Kestler and Kraus showed the limitations of using ML and CL constraints with hierarchical clustering and proposed a method to limit the scope of the constraints [12]. They used `must-link` and `cannot-link` constraints for their problem, but because of their application, needed different sets of constraints for the top half of their dendrogram than for the bottom half. They created an algorithm to do so and conjectured that their work could be extended so that constraints could be specified at more than the two levels. Bade and Nürnberger proposed a new type of constraint for hierarchical clustering, `must-link-before` (`MLB`), a trinary relationship that specifies that two points should be joined together before either are joined to a specified third point in the resulting hierarchy [2, 3]. We show these and other constraints can be naturally modeled in our framework.

Recently there has also been work by Davidson, Ravi, and Shamis which showed how 2-SAT solvers can be used as a basis to form non-hierarchical clustering algorithm's objective functions [8]. This paper instead looks at modeling complex constraints (not the objective function) in hierarchical clustering and using Horn-SAT.

3. A FORMULATION FOR CONSTRAINED HIERARCHICAL CLUSTERING

In this section we outline at a high level how to build a knowledge base that can both model a dendrogram and the constraints the dendrogram must satisfy. The implementation of our constraint solver propositionalizes the knowledge base, so we only add sentences that will propositionalize into Horn clauses. We first define the trinary relationship `together(x,y,i)` which will express that instances x and y must be in the same cluster at level i in the dendrogram. We will then show in the next section four properties that a dendrogram must satisfy and define them using the `together` relationship.

DEFINITION 1. ***`together` Relation.*** *`together(x,y,i)` which if true* **requires** *points x and y to be in the same cluster at level i.*

Our work in this section can be summarized as follows.

- We give a definition of dendrogram using the `together` relationship that requires the following properties to be satisfied: `Transitivity`, `Symmetry`, `Inter-level link` and `Full-tree` (see Section 3.1).
- Each of four properties can be expressed in Horn form (see Section 3.3).
- The definition of dendrogram we use is quite broad and includes dendrograms where no, one, or multiple merges occur at each level in the tree.
- We show that all types of existing constraints can be easily modeled using the `together` relationship and expressed as Horn sentences (see Section 3.2).
- Our formulation lends itself to both new global and local types of constraints.

Once we have created a set of sentences (a knowledge base) to represent our problem and the constraints, a SAT solver will try to find a model that satisfies all of the sentences and thus can be interpreted as a dendrogram that satisfies the user constraints. If it is unable to do so then the set of sentences is unsatisfiable, which would manifest itself in a contradiction. **Our knowledge base is in first order logic but we propositionalize it and use an efficient solver.**

3.1 How to Model a Dendrogram

We now describe a logic formulation for hierarchical clustering that allows us to express each of the previously described constraints in the literature (global, `must-link-before`, level specific) and new variations. The definition of **dendrogram** we use is a sequence of clusterings where each clustering is derived from the previous clustering in the sequence through zero or more merge operations[1]. A **complete dendrogram** is defined as a dendrogram that has only one cluster in the final clustering in that sequence. Throughout this paper, we refer to the "leaf end" of a tree as the first level of a dendrogram, and the root of the tree as the top level of a dendrogram.

We first constrain `together` to reflect a clustering for each value of i by requiring it to respect the equivalence relation properties of transitivity, symmetry, and reflexivity. We then force each clustering at different levels of i to be related through merges by enforcing the inter-level link property. Finally we ensure the relationship will represent a complete dendrogram by enforcing the full-tree dendrogram property. These properties are described formally below. By adding sentences that enforce these properties to our formalism, any satisfying model will be interpretable as a full dendrogram. We will then be able to add our own domain or problem specific constraints to find hierarchical clusterings that satisfy those constraints as well (assuming they are consistent).

Reflexivity.

Every instance is always in the same cluster as itself.

DEFINITION 2. ***Reflexivity Property***

$$\forall i, x[\texttt{together(x,x,i)}]$$

[1]Requiring one or more merges per level, although a more natural formulation, leads to an NP-Hard constraint satisfaction problem.

Symmetry.

If instance a is in the same cluster as instance b at a particular level i, then a is also in the same cluster as b at that level.

DEFINITION 3. ***Symmetry Property.***

$$\forall i, x, y[\texttt{together(x,y,i)} \rightarrow \texttt{together(y,x,i)}]$$

Transitivity.

If instance a and instance b are in the same cluster (`together(a,b,i)`) and instance b and instance c are in the same cluster (`together(b,c,i)`) then instance a and instance c are in the same cluster (`together(a,c,i)`).

DEFINITION 4. ***Transitivity Property.***

$$\forall i, x, y, z[\texttt{together(x,y,i)} \wedge \texttt{together(y,z,i)} \rightarrow \texttt{together(x,z,i)}]$$

Implied Inter-Level Links.

Due to the nature of hierarchical clustering, if two instances are in the same cluster at a given level, they will also be in the same cluster at higher levels in the hierarchy. Another way of putting this, is that once two instances have been merged into the same cluster, they can never be unmerged.

DEFINITION 5. ***Implied Inter-level Links***

$$\forall i, x, y[\texttt{together(x,y,i)} \rightarrow \texttt{together(x,y,i+1)}]$$

Complete Dendrograms.

If our resulting dendrograms are to form a complete tree, then the top level of the hierarchy must contain just one cluster containing all instances.

DEFINITION 6. ***Complete Dendrogram Property***

$$\forall x, y[\texttt{together(x,y,top)}]$$

3.2 Modeling User Constraints

The power of using a logic formulation for hierarchical clustering is that many different types of user constraints can be easily added. All of the constraint types described in previous work (instance constraints [7], level specific instance constraints [12], `must-link-before` [3]) can be formulated using only Horn clauses of our core relation (see Definition 1). We also show how the expressiveness of logic allows us to easily extend the `must-link-before` constraints to encode a notion of distance.

Global constraints.

In previous work Davidson and Ravi explore using `must-link` and `cannot-link` constraints for hierarchical clustering. The `must-link` and `cannot-link` constraints were originally formulated for use with non-hierarchical clustering [7]. Since each level of a dendrogram effectively corresponds to a different non-hierarchical clustering, an obvious interpretation of the constraints is that they must hold in each and every level of the dendrogram. We refer to such unconditional `must-link` and `cannot-link` constraints as global constraints since they apply to each level of the dendrogram. Note that a single `cannot-link` constraint prevents a full dendrogram from being constructed. Therefore the use of the global `cannot-link` constraint cannot be used without removing the full dendrogram requirement.

A global constraint can easily be specified in our framework by replicating the constraint at each level in the dendrogram.

OBSERVATION 1. ***Encoding Global Must/Cannot-Link Constraints [7]***

$$ML(a,b) \equiv \forall i \quad \texttt{together(a,b,i)}$$

$$CL(a,b) \equiv \forall i \quad \neg\texttt{together(a,b,i)}$$

Must Link Before Constraints.

The `must-link-before` constraints formulated and explored in the work of Bade and Nürnberger, are important because they are created specifically for use with hierarchical clustering and are simple to create in practice. `must-link-before` is a trinary relationship that specifies that two instances, a and b, are more similar to each other than either is to a third instance, c. Hence any satisfying dendrogram must have a and b in the same cluster before either is in the same cluster as c. A key observation is that the `must-link-before` constraint is logically equivalent to the claim that if either of a or b are in the same cluster as c, then a and b must already be in the same cluster at the previous level (higher levels are closer to the dendrogram root).

OBSERVATION 2. ***Encoding Must-Link-Before Constraint [3]***

$$\texttt{must-link-before(a,b,c)} \equiv$$

$$\forall i \quad \texttt{together(a,c,i+1)} \rightarrow \texttt{together(a,b,i)}$$

$$\forall i \quad \texttt{together(b,c,i+1)} \rightarrow \texttt{together(a,b,i)}$$

$$\neg\texttt{together(b,c,1)}$$

$$\neg\texttt{together(a,c,1)}$$

The last two sentences are necessary to handle boundary conditions and simply state that the distant point cannot be linked with either of the close points in the first merge.

Level Specific Constraints.

The constraints specified by Kestler and Kraus allowed sets of ML and CL constraints to be specified at different levels of the hierarchy. The $together_i$ relation maps directly to that notion as we can specify ML constraints at a specific level i. To specify cannot links we can use a negated $together_i$ relation which is also in Horn form (because a sentence consisting of a single negative literal meets the definition of a Horn clause, see Definition 7). These constraints have previously been used successfully in specific applications but in general they are not easily generated because domain knowledge of the form that two instances should be together or apart at specific levels in the dendrogram is rare. However, if such information is available it can be both useful and complex to satisfy. Consider if the domain expert states that x and y should be together at level i, a and b together at level $i+1$, and x and a should be apart at level $i+2$. Due to entailment this will generate a range of additional constraints at level $i+2$ namely cannot-links between $(x,b),(y,a),(y,b)$. If the domain expert were to inadvertently state any of these pairs should be together this would create an inconsistency. The benefit of our approach at modeling constraints in a logic and using solvers is that such inconsistency can be easily checked in polynomial time.

Other Constraints.

Our underlying core relation (see Definition 1) can be used to model a variety of constraints. For example, it is easy to extend the MLB constraint to take into account distances between merges. For example, while a constraint $MLB(a, b, c)$ specifies that instance a and b must be merged together before a and c or b and c, it does not say how soon before they should be merged. We may wish to specify that a and b should be merged together much sooner before a or b are merged with c. This may be the case if we believe a and b share a close common ancestor but are only remotely related to c. Consider the situation where a and b should not be merged with c until d merges/time-steps. This can be formulated very similarly to how `must-link-before` constraints are.

OBSERVATION 3. ***Encoding MLB-After-d***

$$must - link - before_d(a, b, c) \equiv$$

$$\forall i \quad \texttt{together(a,c,i+d)} \rightarrow \texttt{together(a,b,i)}$$

$$\forall i \quad \texttt{together(b,c,i+d)} \rightarrow \texttt{together(a,b,i)}$$

$$\neg \texttt{together(b,c,1)}$$

$$\vdots$$

$$\neg \texttt{together(b,c,d)}$$

$$\neg \texttt{together(a,c,1)}$$

$$\vdots$$

$$\neg \texttt{together(a,c,d)}$$

The interpretation of this constraint relies on how merges are performed, which is algorithm dependent. If exactly one merge occurs per level then it will be the case that a and b will be together d merges before either are merged with c. This extended MLB then can be considered as a strong suggestion as to the distance between the triple. In the case when d=1, this definition reduces to the `must-link-before` constraint.

3.3 Propositionalization

Russel and Norvig give a general method for creating a set of set of propositional sentences (CNF) from any first order knowledge base [16]. A satisfying truth assignment for the propositional sentences can be translated into a model satisfying the first order knowledge base. All of the first order logic sentences we used to define a dendrogram (reflexivity, symmetry, transitivity, implied inter-level link) and all user constraints described in this paper can be propositionalized into Horn clauses.

DEFINITION 7. *[Horn Clause] A Horn clause is a disjunction of literals (propositions or negated propositions) in which at most one of the literals is not negated. A Horn clause is logically equivalent to a sentence in which the conjunction of the variables from the negative literals, imply the variable from the only positive literal. For example* $\neg P \vee \neg Q \vee R \equiv (P \wedge Q) \rightarrow R$

The naive approach of model-checking [16] exhaustively tries all combinations of propositional values (true or false) but will take exponential time. The SAT problem is famously known to be NP-complete and therefore no general-purpose polynomial time algorithm is capable of exactly solving it (assuming P$\neq$ NP)[5, 11]. If instead of allowing general logical sentences, we restrict the sentences to be composed only of Horn clauses, then there are efficient algorithms that can solve the satisfiability problem. In this paper we use the algorithm presented by Dowling and Gallier which can either find a minimal satisfying truth assignment, or determine that the sentences are inconsistent in linear time with respect to the total number of literals in all of the Horn clauses [9]. When a satisfying truth assignment is found, it can be used to construct a dendrogram that will satisfy all of the user constraints.

4. ALGORITHMS

In this section we describe our algorithm for incorporating constraint solvers into agglomerative algorithms. As shall be seen, our work is algorithm independent and we experimentally show its performance using single and complete linkage algorithms. It is worthwhile discussing at a high level how our work can be incorporated into the agglomerative algorithm. This is shown in Figure 1.

Algorithm Generate Constrained Dendrogram
Input: $C_{dendrogram}$: The clauses that model the properties of a dendrogram (see section 3.1).
C_{user}: The clauses representing the different types of given user constraints (see section 3.2).
X: A set of data points to build a dendrogram from.
Output: D: a dendrogram that at each level satisfies all constraints.

1. Let $KB = C_{dendrogram} \cup C_{user}$
2. **if** $(RequiredJoins = Solver(KB))$ *fails* **then**
   exit // the user constraints are contradictory

   // Note $RequiredJoins$ is a list of triples $< x, y, i >$.
   // Each triple states that these two points must be
   // joined at this level. These are points that must be joined
   // to satisfy the constraints in KB.

3. **while** (Agglomerative-Algorithm Not Converged)
   a) Join = Agglomerative-Algorithm(RequiredJoins,X)
   // The agglomerative algorithm choose the best legal join
   b) $KB = KB \cup$ Join
   c) RequiredJoins = Inc-Solver(RequiredJoins, KB)
   end
4. Construct D from RequiredJoins

Figure 1: Our Algorithm Used in the Experimental Settings.

Firstly, in Line 1, the clauses representing the user constraints and the dendrogram properties are added to form the core knowledge base. This knowledge base is fed into our solver in Line 2. If the solver fails then the constraints are inconsistent and should be modified (future work will explore using MAXSAT and other formulations to determine which constraints to repair). If the solver succeeds it returns a list of required joins at given levels to satisfy the constraints. Line 3 applies any agglomerative algorithm to select appropriate joins. The solver is called at each level in the dendrogram because the interaction between the joins the algorithm chooses, and the user constraints, may produce additional constraints. In Sections 4.2 and 4.3 we will give the details of our constraint solver. We will give the full details of how the solver fits into a hierarchical clustering algorithm in Section 4.1.

4.1 Full Constrained Hierarchical Clustering Algorithm

Our propositional constraints along with a Horn-SAT solver can be used by any agglomerative hierarchical clustering algorithm. Agglomerative algorithms typically work by finding closest points, merging them, and then updating the distance matrix for the new clusters. This same procedure can be used with our constraint solver to solve constrained hierarchical clustering. When an agglomerative algorithm choses two closest points a and b to be merged, this can be turned into a constraint `together(a,b,i)` where i is the current step the agglomerative algorithm is on. This constraint can be added to the set of logic sentences specifying the constrained problem (the knowledge base). Solving the SAT problem will determine if the merge violates the constraints. If it does violate the constraints, the sentences will be unsatisfiable and the algorithm will need to pick the next closest pair of points until it finds a pair that does not violate the constraints of the problem.

In the case that adding the constraint does not cause the knowledge base to become inconsistent, the Horn-SAT solver will produce a variable assignment which will include the results of merging the two points. In this case the SAT solver need not be restarted, and more merges can continue to be processed without restarting the solver. Merging two clusters may have logical consequences that causes other unrelated clusters to merge together. For example if a constraint `MLB(a,b,c)` is specified, and the merge chosen by the algorithm causes b and c to be put in the same cluster before a and b are in the same cluster, then a logical consequence is that a and b will be merged at a lower level. Therefore, when updating the distance functions, it is generally necessary to check the details of the resulting variable assignment rather that just updating based on the points that were explicitly merged together.

4.2 Horn-SAT

Our Horn-SAT solver determines if a set of Horn clauses is satisfiable, and if so, produces a minimal satisfying truth assignment. A minimal truth assignment will have the fewest number of variables set to true as possible. There are three types of Horn clauses: (1) those consisting only of a positive literal, (2) those with negative literals and a positive literal, (3) and those with only negative literals. In the case when there are no clauses consisting solely of a single positive literal, the clauses are always satisfiable by assigning every variable the value false. When there are no clauses that only have negative literals, then the set of clauses can always be satisfied by assigning every variable the value true. Therefore, the single positive literal clauses are what initially drive the process of requiring variables to be assigned the value true. Setting a variable to true can lead to simplification in the remaining clauses. If all of the negative literals in a clause are set to true, then any positive literal in the clause must be set to true. If such a clause has no positive literal, then the clause has been simplified to the unsatisfiable `false`, and the set of clauses is unsatisfiable.

The Horn-SAT algorithms proposed by Dowling and Gallier [9] works by starting with an all false variable assignment and only assigning a variable the value true if necessary (clauses with only a positive literal, or have been simplified to have only a positive literal). Once a variable is assigned the value true, the variable is added to a queue of variables that will be used to further simplify the remaining clauses. A mapping between each variable and a list of all clauses containing the variable as a negative literal is maintained. If the simplification leads to a clause with a single positive literal, then the variable from the positive literal is added to the queue. The simplification can also lead to the unsatisfiable sentence `false` in which case the set of clauses is unsatisfiable. The algorithm finishes when all variables in the queue have been processed and has a linear runtime with respect to the total number of literals in all of the clauses whose satisfiability is being tested.

4.3 Horn-SAT With Equivalence Relation

The Horn-SAT solver described in the previous section will work for our purposes but will require us to create a large number of propositional sentences to model the properties used to define a dendrogram. The modifications to the solver outlined in this section allow the dendrogram properties to be modeled implicitly, leaving propositional sentences only for use in expressing the user constraints. Our custom Horn-SAT solver takes advantage of our propositions being based on the `together` relationship. Given a constant i, the `together(x,y,i)` relationship satisfies reflexivity, symmetry, and transitivity, making it an equivalence relation. For each possible level in the hierarchy we maintain a disjoint set data structure which can be used to manipulate an equivalence relation. A disjoint set data structure is initialized so that each point is in its own set and the operation of the disjoint set data structure are `merge(x,y)`, which merges the sets that contain x and y, and `find(x)` which returns the name of the set x belongs to. Using the disjoint set data structure, we can set the value of a proposition `together(x,y,i)` to true by selecting the i^{th} disjoint set data structure and executing `merge(x,y)`. Furthermore we can test the value of the same proposition by evaluating `find(x)==find(y)`. Each disjoint set data structure will naturally assure all of the equivalence relation properties of our dendrogram definition are enforced. To ensure that the inter-level link property is enforced, any time a proposition `together(x,y,i)` is set to true the solver will also set `together(x,y,k)` to true for all $k > i$. To enforce the full dendrogram property all of the instances in the top level disjoint-set data structure must be merged into one cluster on initialization.

Our algorithm processes variables in the queue as normal, except whenever a variable needs to be set to true, we simply perform a `merge` operation. However such merges lead to many propositions becoming true (assuming either set has more than one instance). To avoid tracking those propositions that become true during merges, we do not end the algorithm after the positive variable queue becomes empty. Instead, every time the queue becomes empty we scan through the list of variables that are still contained as negative literals in some clause. We check the truth values of such variables, and if they are true, we add them to the queue.

4.4 Runtime Analysis

In this section we give a brief analysis of the general Horn-SAT algorithm, then an analysis of the runtime of our custom solver. For those who initially wish to skip the analysis, the overall runtime to initialize and solve the satisfaction problem with our custom solver is $O({l_n}^2 + n^2)$, where l_n is the total number of negative literals contained in the propositional Horn clauses, and n is the number of instances being clustered. Additional constraints can be added to the knowledge base and incrementally solved for in $O({l_{nr}}^2)$ time where l_{nr} is the number of negative literals in the remaining Horn clauses (calls to the solver generally leads to simplification of the Horn clauses leading to fewer negative literals).

In the general Horn-SAT solver, each variable that must be set to true is put on the queue once. When processing that variable we check which Horn clauses are affected by setting the variable true. Only the clauses that have negative literals that use the same variable as the variable being set to true are affected. Each variable that shows up as a positive literal in any of the clauses, has an associated list of clauses, where each clause on the list is a clause

that will be affected if the variable is set to true. When a variable is set to true all negative literals using that variable are removed from their clauses (and implicitly replaced by a false constant). If any of the clauses are left with only a positive literal that has not already been set to true, then that variable is added to the queue.

Adding variables to the queue takes constant time. Removing a negative literal takes constant time. Processing a variable from the queue requires removing some negative literals and possibly adding some more variables to the queue. Each variable can only be added to the queue once. Therefore the total worst case running time is $O(l_n + l_p + i_o)$ where l_n is the number of negative literals, l_p is the number of positive literals, and i_o the time needed to initialize the necessary data-structures. Dowling and Gallier show that $O(l_n + l_p + i_o) = O(l_n + l_p)$ and details of the correctness and running time of their algorithm can be found in their paper [9].

Our custom solver does extra work when processing each variable in the queue. When a proposition `together(x,y,i)` is set to true we merge the points x, y in the disjoint set data structures corresponding to level i and above. A disjoint set data structure can represent multiple disjoint sets and perform find and join operations that for practical purposes run in constant time [17] (actually uses $O(\alpha(n))$ where $\alpha(n)$ is the inverse of the Ackerman function which grows very slowly, e.g. $\alpha(9876!) = 5$). A dendrogram can have at most n levels so a there are at most n constant time merges performed to process a propositional variable in the positive literal queue. Additionally, each disjoint set data structure can be initialized in linear time with respect to the number of instances, leading to an additional $O(n^2)$ initialization runtime cost.

The custom solver also does extra work every time the positive literal queue is emptied. Every time the queue empties, it is necessary to scan through all negative literals still contained in a Horn clause and check if any of the associated variables have become true. In the worst case only one negative literal will be removed from the clause list each time the queue is emptied. In this case the total amount of work added by the necessity to check those variables is $O({l_n}^2\alpha(n)) \approx O({l_n}^2)$. So the total runtime for our custom Horn-SAT algorithm to take into account this extra work is $O((l_n + l_p + {l_n}^2)\alpha(n) + n^2) \approx O({l_n}^2 + n^2)$. After initially solving the satisfaction problem for a set of constraints, adding an additional variable to the positive literal queue to create additional merges as described in Section 4.1, will take $O({l_{nr}}^2\alpha(n)) \approx O({l_{nr}}^2)$ time, where l_{nr} is the number of remaining negative literals.

5. EXPERIMENTS

Previous work has shown that constraints can be advantageous in increasing the quality of clusterings [3, 2, 7, 6, 12]. Therefore in our experimental results we focus on two different questions:

- Does our method of incorporating constraints into hierarchical algorithms (see Figure 1) produce undesirable results?
- Can the more complex constraints such as those that allow incorporating partial dendrograms produce better results?

The first question we wanted our experiments to answer is how adding constraints using our algorithm would increase the quality of the resulting dendrogram. This is an important question, because we would like users to be able to add this functionality into their clustering algorithms and immediately see better results without having to understand the internals of the constraint solver. We therefore devised experiments to see if this approach would yield good results.

The second question is most important and attempts to address how the quality of resulting dendrograms would be affected by using new constraints that our formulation allows, such as the distance encoded `must-link-before`. On the surface our extension to the `must-link-before` constraint certainly appears stronger, but if it does not lead to better results, then the added complexity is not worthwhile. We created some experiments that would directly compare the results of using `must-link-before` constraints, and our extension to `must-link-before` on the same data sets, to determine whether there is a significant difference between the two in terms of dendrogram quality.

5.1 Simulated User Driven Constraints

Previous work in constrained hierarchical clustering has focused on evaluating sets of randomly generated constraints. In some scenarios that makes sense, but we wanted to create constraints that a user might make after finding unacceptable results in an already attempted hierarchical clustering. We generate these by first creating a dendrogram using our hierarchical clustering algorithm with no constraints, and comparing the output to the known hierarchical structure. We find a set of points that are merged together too soon and create a MLB-After-d constraint to enforce the correct merge order. We then continue the process by rerunning the clustering algorithm with the new constraint, finding the next error, and generating a new constraint based on it.

5.2 Measurements

We used two measurements to calculate the quality of our hierarchical clustering results: F-score and H-Correlation. F-score measure the trade-off between precision and recall and relies only on the labels of each instance. To obtain the F-score we assign each cluster a class based on the majority class membership in the cluster. Each cluster has its F-score calculated and weighted by the number of instances in the cluster. Finally the average F-score over all of the clusters in the hierarchy is used to represent the quality of hierarchical clustering.

H-Correlation is a measurement devised specifically for hierarchical clustering [1]. Rather than using the instance labels, the hierarchical structure is used to measure the quality of the results. Therefore the true hierarchical structure of the data is needed to use this measurement. The measurement iterates over all triples (a,b,c) in the data set and counts the number of times the triples satisfy $MLB(a, b, c)$ in the true hierarchy S_t, the learned hierarchy S_l, and both simultaneously S_b. A higher H-Correlation is desirable and is calculated using:

$$H = \frac{S_b}{S_t + S_l}$$

5.3 Data

Artificial Data Set.

We created a method to artificially generate hierarchical data sets. The data sets generated each have 8 classes whose instances are normally distributed with a mean and variance specific to the class. The method with which those Gaussian parameters are specified is what causes the data to have a hierarchical structure. An initial mean corresponding to the root of the hierarchy is used to calculate two new means which correspond to the two clusters at the next lower level in the tree. Recursively, each of those means is used to randomly generate two new means until there are a total of 8 leaves in the tree. The variance at the root node starts at 1 and is reduced by half at each lower level in the tree. This creates a

hierarchy with 5 levels. The first level is the root and has 1 cluster, the second level has 2 clusters, the third 4, the fourth 8, and the last level has as many clusters as there are instances. Each of the data sets were created with a total of 120 five dimensional instances (15 instances per class).

Newsgroup Data Set.

To test our algorithms on a real world data set we utilized the 20 Newsgroups data set [15]. This is a collection of around 20,000 documents from 20 different newsgroups. A freely available version of this data set has already already been processed into a document-term matrix using the lexing capabilities of Rainbow [14]. The number of terms in the document-term matrix is over 60,000 so we used PCA to reduce the number of dimensions to 100. This data has a natural hierarchical structure based on the way the topics were originally organized. In figure 5.3 the hierarchical relationship between the newsgroups is displayed. The class we used for each document when calculating the F-score is the name of the newsgroup the document was posted in.

- **Computers**
  - Hardware
    - comp.os.mswindows.misc
    - comp.windows.x
    - comp.graphics
  - Software
    - comp.sys.ibm.pc.hardware
    - comp.sys.mac.hardware
- **Recreation**
  - Automobiles
    - rec.autos
    - rec.motorcycles
  - Sports
    - rec.sport.baseball
    - rec.sport.hockey
- **Sale**
    - misc.forsale
- **Science**
    - sci.med
  - Technology
    - sci.crypt
    - sci.electronics
    - sci.space
- **Politics**
    - talk.politics.guns
  - International
    - talk.politics.mideast
    - talk.politics.misc
- **Philosophy/Religion**
    - alt.atheism
  - Theism
    - talk.religion.misc
    - soc.religion.christian

Figure 2: Hierarchical structure of the 20 Newsgroup data set.

5.4 Experimental Setup

The experiments we ran attempted to simulate the process users might use to incrementally create constraints for their problems. In our experiments we first perform unconstrained hierarchical clustering and then find triples of instances that were merged in the wrong order in relationship to the known true hierarchy. We take one such triple, create a constraint that will correct it, and run our constrained hierarchical clustering algorithm with the new constraint. We then continue the process, always creating one new constraint based on an erroneous merge order in the previous clustering, and then add the constraint to those from the previous steps and re-cluster. We use standard agglomerative hierarchical clustering algorithms and use the Horn-SAT solver to ensure that the constraints are satisfied. We used both single linkage and complete linkage distance update steps in order to compare how they each behave in a constrained clustering environment.

In order to show the significance of our results we repeated each experiment 10 times. For the artificial data set each independent run of experiments started by randomly generating a new data set. The Newsgroup data set was not randomly generated, but is sufficiently large so that we were able to randomly sample the instances so that each set of experiments had a sample of 200 instances with equal number of instances in each class. After creating the data set, each experiment was executed using the same data set, so the different methods could be compared on the same data. We performed a full combinatorial experiment on: choice of data set, type of constraints and distance function. The results of each of these experiments averaged over all 10 experimental runs are plotted below.

5.5 Results

Our results answered our questions fairly conclusively. We first wanted to determine if using our Horn-SAT solver with standard agglomerative clustering algorithms would improve the quality of our results. The left most measurement on each graph in our results is the measurement taken for unconstrained hierarchical clustering. By adding our `MLB-after-d` constraints we saw improved performance as the number of constraints increased. The improvement is significantly larger than global constraints presented in our earlier work [7] given the constraint is more expressive and capable of encoding a partial dendrogram.

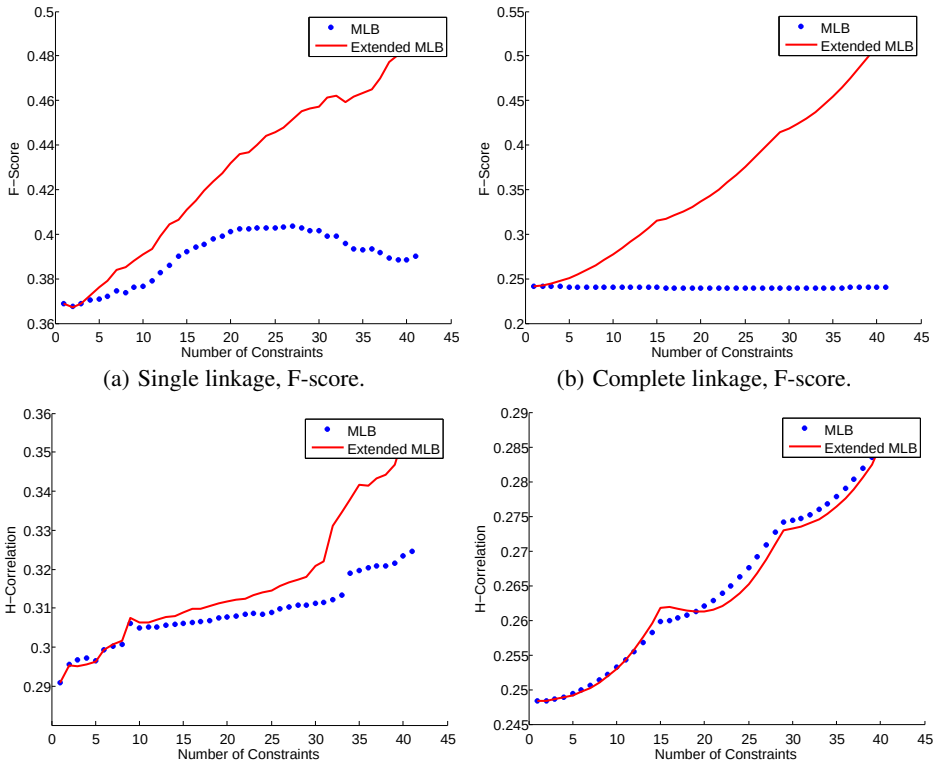
Secondly we wanted to determine if there were advantages to being able to formulate new types of constraints using logic. The F-score measurements in all of the experiments suggest that this is the case since the `MLB-after-d` constraints all outperformed the original `must-link-before` constraints. The H-correlation measurement was created by the inventors of the `must-link-before` constraint and centrally uses that constraint. It is therefore quite encouraging to see that in our experiments using H-correlation, that the MLB After-d constraints were often able to perform better than the `must-link-before` constraint which the measure is based on.

6. CONCLUSION

In this work we attempted to create a constraint solver that would allow any agglomerative hierarchical clustering algorithm to become a *constrained* hierarchical clustering algorithm, through declarative means, rather than procedurally. Logic is a good choice for constraint specification, because it is formal, yet widely understood and studied. There are also many algorithms for solving the satisfiability/consistency problem that can be utilized by a constraint solver.

We showed how logic can be used to model dendrograms and user constraints in such a way that the satisfiability could be tested in polynomial time. All of the constraint types in previous work on constrained hierarchical clustering are expressible in Horn form, as are the logical sentences needed to model a dendrogram. This ensures that we can find solutions that satisfy the constraints efficiently. A still unexplored direction for our framework is maximizing the number of constraints satisfied in the case of an inconsistent set of constraints.

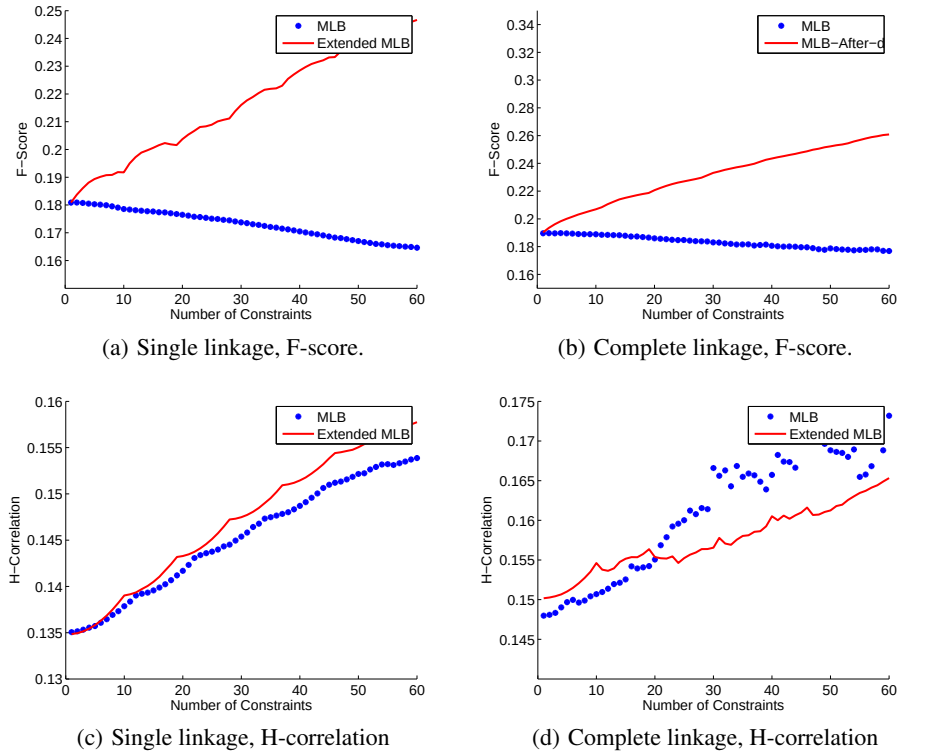
We implemented our solver and integrated it into an agglomerative hierarchical clustering algorithm, with as little extra work as possible. We have also made our solver and experiment code available on-line[10]. Our experiments show that taking just that simple step can lead to a well performing constrained hierarchical clustering algorithm. There are other ways this solver could be used to perform hierarchical clustering, but it was more interesting to us to see how it could be used by existing clustering algorithms with no additional work.



(a) Single linkage, F-score. (b) Complete linkage, F-score.

(c) Single linkage, H-correlation (d) Complete linkage, H-correlation

Figure 3: Artificial data set results.



(a) Single linkage, F-score. (b) Complete linkage, F-score.

(c) Single linkage, H-correlation (d) Complete linkage, H-correlation

Figure 4: 20 Newsgroups data set results.

7. REFERENCES

[1] K. Bade and D. Benz. Evaluation strategies for learning algorithms of hierarchies. In *Proceedings of the 32nd Annual Conference of the German Classification Society (GfKl'08)*, 2009.

[2] K. Bade and A. Nürnberger. Personalized hierarchical clustering. In *WI '06: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 181–187, Washington, DC, USA, 2006. IEEE Computer Society.

[3] K. Bade and A. Nürnberger. Creating a cluster hierarchy under constraints of a partially known hierarchy. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2008*, pages 13–24, 2008.

[4] S. Baso, I. Davidson, and K. L. Wagstaff, editors. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. CRC Press, 2009.

[5] S. A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.

[6] I. Davidson and S. S. Ravi. Agglomerative hierarchical clustering with constraints: Theoretical and empirical results. In *Knowledge Discovery in Databases: PKDD 2005, 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 59–70, 2005.

[7] I. Davidson and S. S. Ravi. Using instance-level constraints in agglomerative hierarchical clustering: theoretical and empirical results. *Data Min. Knowl. Discov.*, 18(2):257–282, 2009.

[8] S. Davidson, Ravi. A sat-based framework for efficient constrained clustering. In *Siam Data Mining (SDM)*, 2010.

[9] W. F. Dowling and J. J. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. Logic Programming*, 3:267–284, 1984.

[10] S. Gilpin. Code for algorithms in this paper. `http://csiflabs.cs.ucdavis.edu/~sagilpin/`.

[11] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[12] H. A. Kestler, J. M. Kraus, G. Palm, and F. Schwenker. On the effects of constraints in semi-supervised hierarchical clustering. In *Artificial Neural Networks in Pattern Recognition, Second IAPR Workshop, ANNPR 2006*, pages 57–66, 2006.

[13] J. R. Kettenring. A perspective on cluster analysis. *Stat. Anal. Data Min.*, 1(1):52–53, 2008.

[14] A. K. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. `http://www.cs.cmu.edu/~mccallum/bow`, 1996.

[15] J. Rennie. 20 newsgroups data set.

[16] S. J. Russell and P. Norvig. *Artificial Intelligence a Modern Approach*. Prentice Hall, Upper Saddle River, N.J., 2nd international edition edition, 2003.

[17] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, 1975.

[18] K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, pages 1103–1110, 2000.