CSC 793 report

Evaluation of Axiom Selection Techniques

Student: Zihao Wang

Supervisor: Geoff Sutcliffe

University of Miami, USA, [zxw526@miami.edu](mailto:zxw526@miami.edu), geoff@cs.miami.edu

# Abstract

Evaluation of Axiom Selection Techniques

# Introduction

“Large Theory” problems in Automated Theorem Proving (ATP) have been defined as having *many functors and predicates, and many axioms of which only a few are required for the proof of a theorem*. Large theory problems are mostly found in corpora that contain very many problems, e.g., the MPTP2078 corpus [**?**], the Mizar 40 corpus [**?**], and the GRUNGE corpus [**?**]. Large theory problems present challenges to ATP systems, because of the large search space generated by the large number of axioms. One key to solving large theory problems is selecting a subset of the axioms that is adequate for finding a proof. There has been significant and successful research on this topic, e.g., [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**]. Many techniques are based on the occurrences of symbols in the formulae, e.g., the SInE method [**?**] and its derivatives. The fact that large theory problems often occur in large corpora makes the application of machine learning techniques [**?**] viable, e.g., as done in the MaLARea system [**?**].

Evaluation of axiom selection techniques is typically done by:

* 1. Choosing a corpus of large theory problems.
  2. For each problem in the corpus, selecting a subset of the axioms.
  3. Running an ATP system on a reduced problem formed from the selected axioms and the problem’s conjecture.

When such experiments are done on large corpora it is necessary to impose a small time limit when running the ATP system. In the last step of this proces a proof indicates an adequate selection, and a countermodel indicates an inadequate selection. A timeout provides no infor- mation - the selection might be inadequate, or the selection might be adequate but the reduced problem is too hard because too many (unnecessary) axioms were selected and the time limit is too small. The results are also influenced by the choice of ATP system.

This paper presents metrics for evaluating axiom selection techniques without having to run an ATP system on the reduced problems. While the “proof is in the pudding” and it is eventually necessary to evaluate by running an ATP system, the method described in this paper provides a first-pass evaluation that allows axiom selection techniques to be rapidly tested and refined. The approach has the advantage of being independent of a chosen ATP system.

This paper additionally presents some new axiom selection techniques. The new techniques the axiom selection built into the Vampire [**?**] and E [**?**] ATP systems, are evaluated using the proposed metrics.

This paper is structured as follows: Section **??** describes the evaluation metrics. Section **??** describes a distance measure between formulae, and some new axiom selection techniques based on that measure. Section **??** provudes evaluation results. Section **??** concludes.

# Selection Metrics

The principle behind the new evaluation metrics is to compare the selected subsets of axioms with known adequate sets of axioms. In this work the MPTP2078 corpus has been used. The MPTP2078 corpus has two versions of each of the 2078 problems: the *bushy* (small) versions contain only the Mizar axioms that are known to be directly used in the proof of the conjecture, and the *chainy* versions contain all the axioms that precede the conjecture in the Mizar library order. The bushy problems contain between N and M axioms, while the chainy versions contain between N and M axioms.

In order to extract known adequate sets of axioms for each problem, Vampire and E were run on the problems with a 300s CPU limit. This produced proofs for NNNN of the bushy problems (KKK by Vampire and JJJ by E) and MMMM of the chainy problems (KKK by Vampire and JJJ by E). For the problems solved, the axioms used in proofs were extracted as known adequate sets of axioms, and new problems formed from those adequate sets with the corresponding conjectures. Additionally, in testing the new axiom selection techniques some further different adequate sets were found, and further new problems created. This resulted in new problems for NNNN of the bushy problems and MMMM of the chainy problems. For some problems multiple adequate sets of axioms were found, resulting in a total of GGGG new problems for the NNNN bushy problems and a total of HHHH new problems for the MMMM chainy problems. The new problems based on the adequate sets have been been used to augment the MPTP2078 corpus, as the *Pruney* problems. The pruney problems provide adequate known sets of axioms against which selected sets of axioms can be compared.

GEOFF: Description of metrics. Section on selection techniques - cuttion (eg Isabelle) and projection (eg SInE).

# Selection Techniques

## Intro

## The Symmetric Weighted-average Extended Hutchinson Distance

Terms are the most basic structure in the first-order logic. A reasonable term metric used for evaluating the term difference can guarantee its extended atom and formula metrics perform well to some extent. For two arbitrary terms *t*1, *t*2, suppose that there exists a substitution *θ* such that *t*1*θ* = *t*2, which reflects the changes from *t*1 to *t*2. In fact, not all two arbitrary terms exist a substitution, and it is thus unrealistic to evaluate the term difference by substitutions directly. To address the problem, we take the least general generalization *lgg* of terms as a medium to evaluate the term difference. Besides, we also propose a way to evaluate the term functional and variable difference separately.

Let θ = {X1 1→ u1, ..., Xn 1→ un, Y1 1→ Z1, ..., Ym 1→ Zm} be an arbitrary substitution, where u1, ..., un are functional terms, Z1, ..., Zm are variables, and X1, ..., Xn, Y1, ..., Ym are distinct variables. functional substitution θf and variable substitution θv are defined as:

*θf* = *{X*1 *1→ u*1*, ..., Xn 1→ un}*,

*θv* = *{Y*1 *1→ Z*1*, ..., Ym 1→ Zm}*.

Obviously, θf ∩ θv = ∅ and θf ∪ θv = θ. In some special cases, θf or θv may be empty substitution. We then put forward two functions Sf and Sv, mapping θf and θv to real numbers, in which Sf mainly considers the functional difference while Sv only takes the variable difference into account. Sf , Sv are defined as:

*n*

*Sf* (*θf* ) = *occt*1 (*Xi*) *×* (

*w*(*f* ) *× occui* (*f* )); (1)

*i*=1

*m*

*f∈F* (*ui*)

*Sv*(*θv*) = *w*0 *×* (*log*(*occ*+ (*Zj*) *− occ*+ (*Yj*)))*.* (2)

*j*=1

*t*2

*t*1

Where,

* + - *w* is a weight function that maps every function symbol to a non-negative integer;
    - *w*0 (*w*0 *>* 0) is a constant representing the weight for every variable;
    - *F* (*t*) denotes the set of all function symbols appearing in a term *t*;
    - *occt*(*s*) denotes the number of occurrences of a function or variable symbol *s* in a term *t*;

*occt* (*v*) denotes the number of deep occurrences of a variable *v* in a term *t*, which takes the depth of *v* (the number of symbols nested *v*) into consideration. For every occurrence

+

*•*

+ *occt*(*v*)

*i* of *v*, the depth of *v* is *ni* (*ni ≥* 0). *occ* (*v*) = *ni* + *occt*(*v*).

*t*

*i*=1

Given two terms *t*1 and *t*2, *t* is their the least general generalization with substitutions *θ*1 and *θ*2, such that *tθ*1 = *t*1 and *tθ*2 = *t*2. Based on proposed functions, the term difference function *dT* between *t*1 and *t*2 is defined as:

*dT* (*t*1*, t*2) = ✓[*Sf* (*θ*1*f* ) + *Sf* (*θ*2*f* )]2 + [*Sv*(*θ*1*v* ) + *Sv*(*θ*2*v* )]2*.* (3)

Compatible atoms can also construct the most general generalization as terms do in the same way, thus the dT can apply to them naturally. As for incompatible atoms, we simply think their difference is extremely huge due to they cannot be unified. Hence,

* + - if *A*1, *A*2 are compatible atoms, *dA*(*A*1*, A*2) = *dT* (*A*1*, A*2);
    - if *A*1, *A*2 are incompatible atoms, *dA*(*A*1*, A*2) = +*∞*.

First-order formulas are the connection of atoms, logical connectives and quantifiers. Ignor- ing all the logical connectives and quantifiers, formulas are sets of atoms. The inference will not happen in such formulas, if all atoms in them are incompatible. We assert that the more incompatible atoms two formulas have, the less similar the formulas are.

Suppose that F1, F2 are two formulas. Let D1 = {A1, ..., An} and D2 = {B1, ..., Bm}, which denote the corresponding atom sets, respectively. Penatly = |{dA(Ai, Bj)|dA(Ai, Bj) =

(+∞, +∞)}| is the number of incompatible atom pairs (Ai, Bj) in formulas. The formula difference function dF between F1 and F2 is defined as:

*n m*

*d* (*F , F* ) = *n × m*

*F*

1

2

*Pentaly*

*A*

*i*

*j*

*× d*

*i*=1 *j*=1

(*A , B* ) (4)

First, we converted a logic problem into a fully-connected graph G = (V, E), where each node vi represents a logical formulae, and each edge eij has weight witj representing the dissimilarity between nodes vi and vj. We devised the following method to convert each dissimilarity weight witj into a similarity weight wij:

*wij* = max(0*,* arg max(*witj*

*wiIj ⊂W*

*∞*) *− witj* ) (5)

The conversion method consists of three cases:

* + - If the dissimilarity weight *witj* is equal to the largest dissimilarity weight that is not infinity, or if *witj* = *∞*, then our method will assign a similarity weight *wij* = 0 to the edge *eij*.
    - If the dissimilarity weight *witj* = 0, then our method will assign a similarity weight equal

to the largest dissimilarity weight that is not infinity (i.e. arg max*wiIj ⊂W* (*witj ∞*)).

* + - Any dissimilarity weight *witj* between these two extremes will yield a similarity weight *wij*

that is between 0 and arg max*wiIj ⊂W* (*witj* ).

We used the method to convert the dissimilarity matrix *Wt* into a similarity matrix *W* .

## Axiom selection algorithms design

In this chapter, 3 methods of axiom selection based on the graph was proposed. First, using spectral clustering to select about half of the needed axioms. Second, based on the first step, using local search to find path from the conjecture to one of it’s infinity distance axiom. The local search algorithm is a greedy strategy by finding the optimal solution only based on the current state. There are local beam search algorithm is implemented. There are 2 experiments. The first one is finding a random path which each node is visited its nearest neighbor. The process starts from conjecture to one of the axiom that is the infinity distance from the conjecture to them. And second one is find all paths from the conjecture to infinity distance node connected to the conjecture. After step 2 and 3, a spectral cluster algorithm which include more clusters is implemented. Based on the path from the step 2 and 3, each path node’s cluster set node will be added to the path set.

### **Spectral clustering binary classification**

The graph will be divided into 2 clusters by using spectral clustering. Most of the partitions can include all need axioms in one cluster.

Based on the graph weighted by the dissimilarity, a new graph with the same amount of vertices and edges are generated by weighted by similarity by using the Gaussian Radial basis function kernel showed below:

S=e-D

In the formula, S is the similarity matrix, D is the dissimilarity matrix. After generating the similarity matrix, the spectral clustering is implemented by using python sklearn based on the similarity matrix.

The algorithm of spectral clustering of sklearn is illustrated below:

input: similarity

output: an array construct of integers which means each element’s cluser

Step 1: Generate degree matrix D from the similarity matrix

Step 2: Generate Laplace matrix L

Step 3: Calculate normalized Laplace matrix L1 by using Ncut algorithm

Step 4: Calculate L1’s eigenvalue and each eigenvalue’s eigenvector f

Step 5: Use k-means based on the normalized eigenvector

The steps illustrated above are packaged in the sklearn library. After spectral clustering, the majority automated theorem proof (ATP) task can separate in to 2 parts which has 1 part includes all the needed axioms.

### **Local beam search for axioms selection**

The local beam search uses greedy search strategy which is find the local optimal state based on the current state. However, the local beam search not only find one local optimal state, but also keeps track more local optimal states. In this study, the definition of the local optimal solution are the minimum dissimilarity axiom nodes connected to the current node. The local beam search is implemented based on the breadth first search (BFS).

In this study, the BFS only considers the neighbors which has the minimum dissimilarity. The BFS start at the conjecture and finish once reach an infinity dissimilarity node of the conjecture (terminate node). There are 2 experiments. The first is randomly find a minimum dissimilarity node if there are more than 1 local optimal solutions. The second is search all minimum dissimilarity nodes for all local optimal solutions.

#### **Local beam search random greedy path**

In the first experiment, BFS started at the conjecture and stop when reach an infinity dissimilarity node connected with the conjecture. A FIFO queue is implemented for the BFS search. The pseudocode is illustrated below.



19 path.append(u)

In the pseudocode, s is the start vertex. White means the node haven’t be visited yet. Gray means the node is been visiting now. Black means the node’s all neighbors have been visited. The attribute d means the level of the vertex. Lines 1–4 mark every vertex except the start vertex, set u.d to be infinity for each vertex u, and set the parent of every vertex to be NIL. Line 5 paints s gray,. Line 6 initializes s.d to 0, and line 7 sets the predecessor of the source to be NIL. Lines 8–9 initialize Q to the queue containing just the vertex s.

The randomness happened due to the order of enqueue is random. After dequeue the first node u, its child v will recode its parent to be u. After an terminate node is visited. The backtrack process will be implement by recursively search node and its parent. The path is saved in a set without consider the order.

#### **Local beam search all greedy path**

The second experiment is finding all greedy path if one node has more than 1 minimum dissimilarity neighbors. The same BFS process is used. If the dequeue vertex u is a terminate node, then it’s neighbor will not visited and mark u’s color to black and add to the path set. The BFS continue until the queue is empty. The process is shown below:



if u != terminate node:



if u = terminate node:



path.append(u)



path.append(u)

The example of search path is illustrated in the Figure 1.

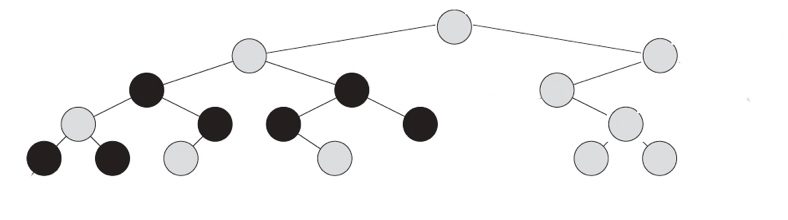


Figure 1: local beam search path

The root represents the conjecture and the black nodes represent the terminate node.

### **A\* search algorithm**

The A\* search algorithm is a kind of heuristic algorithm that find the shortest path from the start node to the goal node. It uses evaluate function f(x) to choose the next step by considering the minimum value of f(x).

The evaluate function f(x) is defined as: f(x)=g(x)+h(x). g(x) is the sum weights of visited edges at the current state and h(x) is the heuristic function which calculate the distance from the next selection to the goal state.

In the axiom selection task. The start state is the conjecture and goal states are a set of all terminate nodes. The task will find the short path from the conjecture to each goal state. The selection set is the union of all shortest path above.

The framework is based on the BFS. The neighbor is selected by using f(x). If there are more minimum f(x) value of the current state, then there are more neighbor are considered to search by using BFS.

### **Deep spectral clustering to improve robustness**

Based on the axiom path set, a deep spectral clustering by using amortized analysis implemented to improve the selectivity of the axiom selection. The amortized idea is that suppose one node is connected with k neighbors which are necessary to prove the conjecture. In this study, k equal to 5 defined by randomness. The spectral clustering’s number of clusters is defined by total number of nodes in subgraph divided by k. For each node in the path, it’s cluster will be found and the neighbors will be added to the path set.

### **K-nearest neighbor algorithm to improve robustness**

The K-nearest neighbor (KNN) algorithm is one of the machine learning algorithms of classification. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors, details illustrated in Figure 2.



Figure 2: example of KNN

In Figure 2, the green point need to be classified into blue or red. In this case, k=3. The green dot finds its closest 3 neighbors. In the 3 neighbors, the majority number of the category is the green dot’s category.

In axiom selection task, the KNN is used to add more axioms to the path of Local beam search all greedy path. For each axiom in the all path set, finding all neighbors and its unique dissimilarity set (UDS). The UDS is defined below:

The UDS is ordered by non-decrease order. There are 3 kinds of UDS classified by the size of less than 3, and greater than 3 or equal to 3. If size(UDS)>=3, then this axiom has the ability to add more axioms because the first element of UDS is 0 which means the dissimilarity of this axiom itself; the second element represents the closest neighbor that are added in the local beam search; the third element represents the second closets neighbor, and so on.

In KNN algorithm, k=5. A subset of UDS (SUDS) is generated by removing the first and the second element. The algorithm is illustrated as follows:

If size(SUDS)>=5:

add 5 closest dissimilarity axioms to the Allpath set  
 else if size(SUDS)>=3:

add 1 closest dissimilarity axioms to the Allpath set

# Evaluation Results

Section on evaluation 1. The test set(s)... Should we add tptp based set? 2. The results 3. The conclusions

Data on MPTPTP2078, Number of problems in test set, how selected (proofs, hence already solved, but possibly with axiom selection), numbers of different adequate subsets, average ratio nntp/all

## Results

Two datasets (Bushy and Chainy) from the MPTP 2078 is used to evaluate the algorithm. There are both 325 tasks in Bushy and Chainy dataset. There are 5 features to evaluate algorithm: the selection evaluation (All score), selected enough number (Enuf), selection evaluation on selected enough tasks (Enuf score) , selectivity and enough selectivity (Enuf selectivity). The definitions of the 4 features are illustrated below.

The all score is the average accuracy of successful selected tasks.

accuracy= the number of need axioms/ the number of selected axioms

If there are some needed axioms not in the selected axioms, then accuracy is 0.

Enuf is the number of tasks that the accuracy is not 0.

Enuf score is the average accuracy based on Enuf tasks.

The selectivity is the average of the number of selected axioms/total number of axioms in the task. The enough selectivity is the average of the number of selected axioms/ Enuf.

The evaluation result is illustrated in the Table 1.

Table 1 : evaluation result

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Bushy | | | | | Chainy | | | | |
| Algorithm | All score | Enuf | Enuf score | selectivity | Enuf selectivity | All score | Enuf | Enuf score | selectivity | Enuf selectivity |
| Spectral clustering | 0.39 | 299 | 0.42 | 0.64 | 0.70 | 0.06 | 325 | 0.06 | 0.94 | 0.94 |
| Local beam search random path | 0.33 | 278 | 0.39 | 0.65 | 0.76 | 0.02 | 157 | 0.04 | 0.30 | 0.63 |
| Local beam search all path | 0.36 | 304 | 0.39 | 0.73 | 0.79 | 0.05 | 294 | 0.05 | 0.84 | 0.93 |
| A\* search all path | 0.13 | 154 | 0.28 | 0.38 | 0.79 | 0.02 | 177 | 0.04 | 0.45 | 0.81 |
| Knn search all path | 0.37 | 315 | 0.38 | 0.77 | 0.80 | 0.05 | 313 | 0.05 | 0.93 | 0.96 |

## Discussions

# Conclusion

GEOFF: 1. Future correlate metrics with ptover performance (or do now!)

# References