September 30, 2008

The HAMLET Project: Year 1 Report

Tim Menzies
LCSEE, WVU
tim@menzies.us

About this document: This report describes progress on the HAMLET "anything browser". It updates a prior version of this document, released mid-year. See the *Change Log* on page 3 for a list of differences.

Contents

Change log:

September 30,2008	**page 3** Change log added **page 6** New concept of operations and, on page 8, a new concept of operations graphic **page 12** Overview section §3.2 motivating the use of this tool. **page 17** The example in §4 has been complete overhauled. Previously, our STEP parser was incomplete and we presented an example using JAVA. Now, our STEP parser is functioning and we can present detailed STEP examples. For other information on our new STEP cases studies, see also Figure 25 (on page 39) and the clustering analysis of Figure 23 (on page 36) and Figure 24 (on page 36). **page 20** Our prior distance functions had some nonlinearities that now been fixed. Figure 10 shows a new design (in red) floating near its nearest related concepts (in green). **page 22** §5.1 describes our new generic parsing protocol. The goal of this protocol is to enable the rapid creation of new new parsers for a wide range of technical document types. **page 30** HAMLET now contains three working tools. §5.4 describes the K-Means, canopy, and GENIC clustering algorithms. **page 36 and 36** We are very pleased with the clustering results reported here. Our new clustering algorithm (GENIC) runs hundreds to thousands of times faster than conventional methods (K-MEANS). In theory, we now have the tool kit that will scale (and this optimism will be tested in future reports).
August 18,2008	Preliminary progress report released

List of Figures

1 Summary

WVU's work into long-term document retention divides into immediate and long-term investigations:

- The immediate concern is for an assessment of the STEP/EXPRESS language for handling long-term data retention. For information on the immediate work, see the reports from Victor Mucino.
- Other, more long-term work, is exploratory in nature. This paper is about HAMLET, which is one example of the exploratory work.

HAMLET is an "anything browser" that aims to offer advice on what engineering parts are relevant to a partially complete current design.

HAMLET makes minimal assumptions about the nature of the engineering documents being explored and is designed to be "glue" that permits the searching of technical data in large heterogeneous collections.

Using technology from AI, information retrieval, program comprehension, and text mining, HAMLET allows designers to dig up prior designs, study those designs, and apply any learned insights to new tasks.

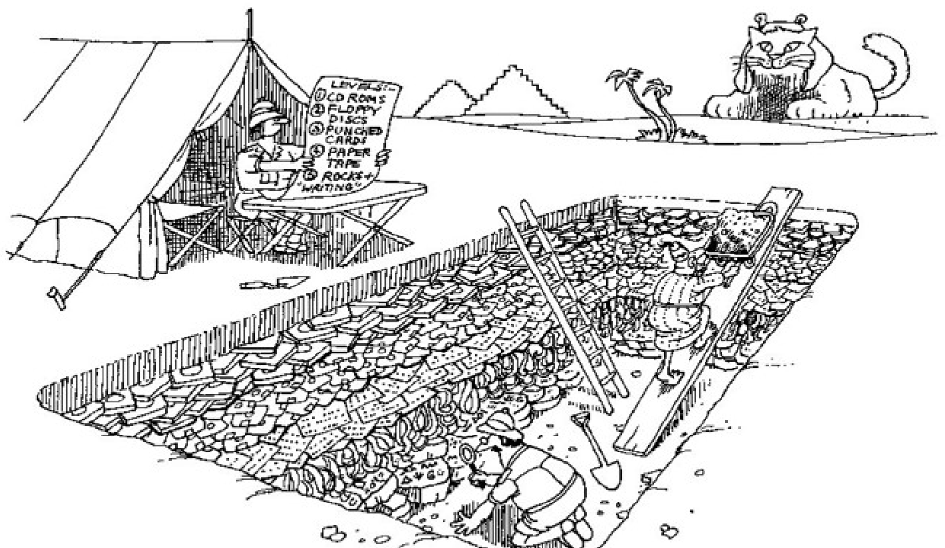


Figure 1: Digging up old designs. From [4].

2 What is HAMLET? (overview)

2.1 A New View on HAMLET

HAMLET was originally conceived as a single application. This "anything browser" was designed to glue together and search technical data in large heterogeneous collections. To achieve this goal, HAMLET offers the following operations:

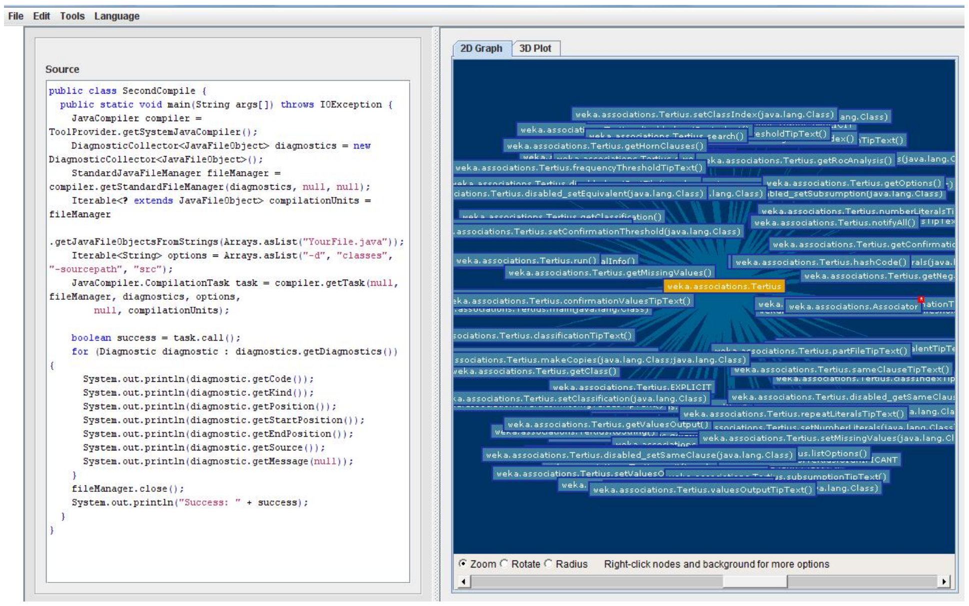
- Provides an import facility from a variety of artifacts. The generic parsing facility is designed to be extensible so, in theory, it would be possible to extend the range of artifacts processed by HAMLET.
- Provides standard visualization techniques for those artifacts; e.g. fast graphical browsing of used and used-by links. While state-of-the-art, these standard visualization techniques quickly overload and exhaust the user (for example, see Figure 2). Hence, HAMLET uses AI, text mining, program comprehension techniques, and information retrieval techniques to offer a set of filtered views on the data.

HAMLET's concept of operation has since changed. It has been realized that the above operations can support a wide variety of activities:

- Originally, we were targeting a "what else" and "what not" design agent that found the delta between a partial description of a current design to other designs.
- Now, we view HAMLET as a reconfigurable library of technical specification comprehension tools. This library supports "what else" and "what not" but could be made to support many other specification comprehension applications such as concept location, validation, etc.

Figure 3 shows the new concept of operations. The application previously known as HAMLET is built on top of a large library of other tools that can be mixed and matched as required.

This new view of HAMLET is very recent and we have yet to take advantage of this new HAMLET-as-library architecture. Future versions of this document will try to exploit this new view. The rest of this document focuses on the use of this architecture for our standard "what else" / "what not" task.


File
Edit
Tools
Language
Source
2D Graph
3D Plot
weka.associations.Tertius
weka.associations.Tertius.setClassIndex(java.lang.Class)
weka.associations.Tertius.getHornClauses()
weka.associations.Tertius.frequencyThresholdTipText()
weka.associations.Tertius.getRocAnalysis()
weka.associations.Tertius.getOptions()
weka.associations.Tertius.notifyAll()
weka.associations.Tertius.run()
weka.associations.Tertius.getMissingValues()
weka.associations.Tertius.hashCode()
weka.associations.Tertius.getClass()
weka.associations.Tertius.EXPLICIT
weka.associations.Tertius.sameClauseTipText()
weka.associations.Tertius.disabled_getSameClau
weka.associations.Tertius.repeatLiteralsTipText()
weka.associations.Tertius.getValuesOutput()
weka.associations.Tertius.disabled_setSameClause(java.lang.Class)
weka.associations.Tertius.valuesOutputTipText()
Zoom
Rotate
Radius
Right-click nodes and background for more options

```
public class SecondCompile {
  public static void main(String args[]) throws IOException {
    JavaCompiler compiler =
ToolProvider.getSystemJavaCompiler();
    DiagnosticCollector<JavaFileObject> diagnostics = new
DiagnosticCollector<JavaFileObject>();
    StandardJavaFileManager fileManager =
compiler.getStandardFileManager(diagnostics, null, null);
    Iterable<? extends JavaFileObject> compilationUnits =
fileManager

.getJavaFileObjectsFromStrings(Arrays.asList("YourFile.java"));
    Iterable<String> options = Arrays.asList("-d", "classes",
"-sourcepath", "src");
    JavaCompiler.CompilationTask task = compiler.getTask(null,
fileManager, diagnostics, options,
        null, compilationUnits);

    boolean success = task.call();
    for (Diagnostic diagnostic : diagnostics.getDiagnostics())
{
      System.out.println(diagnostic.getCode());
      System.out.println(diagnostic.getKind());
      System.out.println(diagnostic.getPosition());
      System.out.println(diagnostic.getStartPosition());
      System.out.println(diagnostic.getEndPosition());
      System.out.println(diagnostic.getSource());
      System.out.println(diagnostic.getMessage(null));
    }
    fileManager.close();
    System.out.println("Success: " + success);
  }
}
```

Figure 2: Too many neighboring concepts: note the overdose of information resulting from a simple visualization of the query "what terms are used by the code in the left-hand-side screen?" This figure shows the terms referenced by the code sample on the left-hand-side to a depth of eight links (i.e. directly references, referenced by something referenced in the left-hand-side code, referenced by something referenced by something referenced in the left-hand-side code and so on to a nested depth of maximum eight links). The depth of nesting for the query is controlled by the slider at bottom right. The code snippet is in JAVA but a similar query on an EXPRESS schema would result in a similar information overload. The lesson of this figure is that special tools are required: we should not show *all* the nested references; just the *relevant* ones. HAMLET's display of relevant local information is shown in Figure 4.

.

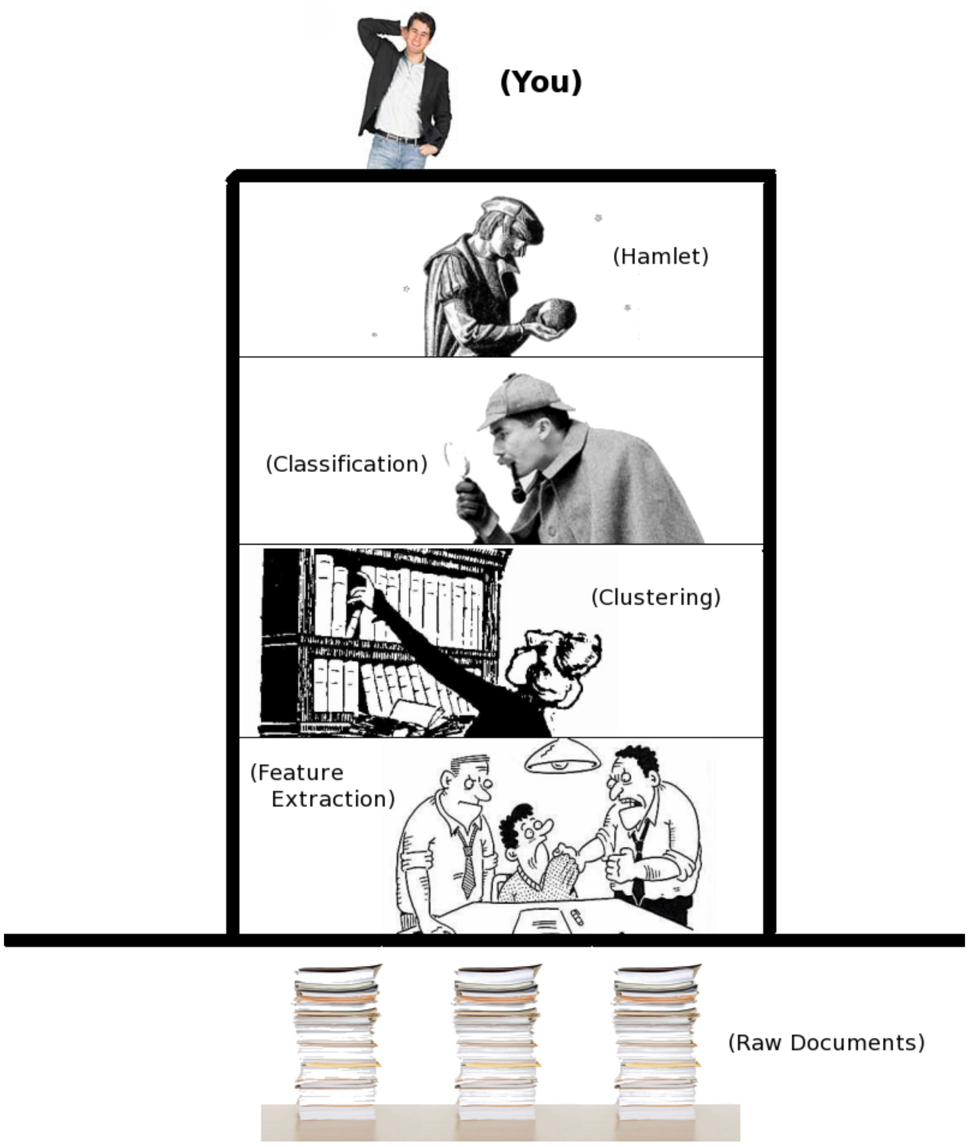


Figure 3: Our toolkit is a set of libraries, on top of which we can implement multiple user-level tools.

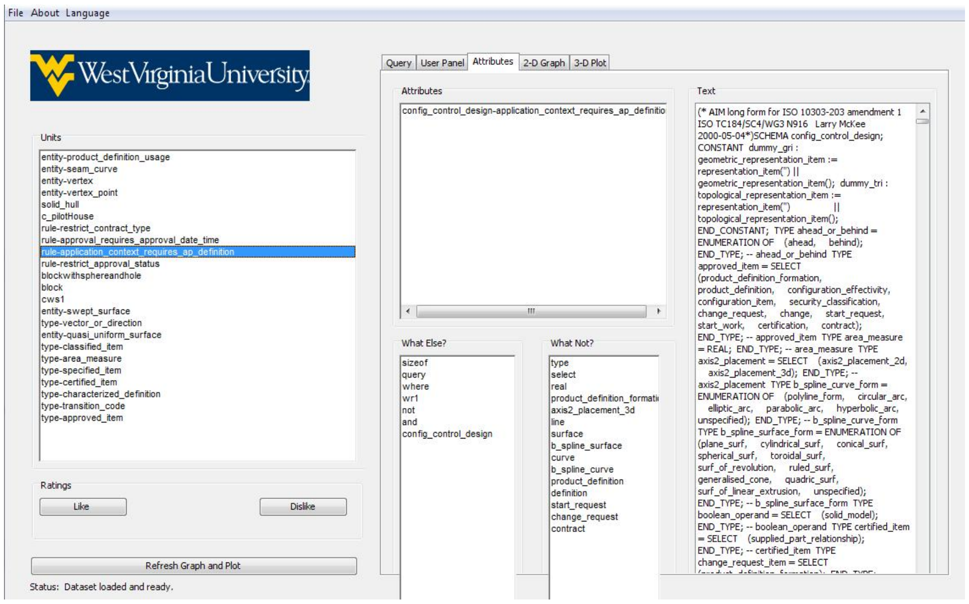
2.2 What Else, What Not

HAMLET was named after the famous Shakespeare quote “to be or not to be?” For the designer of an technical product, the analogoous question is “to do or not to do?” HAMLET finds the deltas between the current design and old design to compute:

- “What else”: what is *absent* from the current design but is usually *present* in older designs.
- “What not”; what is *present* in the current design but is usually *absent* in older designs.

Figure 4 offers an example of these two lists. Note that designers are not obliged to always add the “what else” results or always avoid the “what not” results. However, those two queries will allow a designer to assess their current design with respect to the space of prior designs.

Once HAMLET returns these lists, the designer and HAMLET enter a feedback loop where the designer reviews HAMLET’s “what else” and “what not” list. HAMLET learns the designer’s preferences and, subsequently, uses that knowledge to offer



Units

```
entity-product_definition_usage
entity-seam_curve
entity-vertex
entity-vertex_point
solid_hull
c_pilotHouse
rule-restrict_contract_type
rule-approval_requires_approval_date_time
rule-application_context_requires_ap_definition
rule-restrict_approval_status
blockwithsphereandhole
block
cws1
entity-swept_surface
type-vector_or_direction
entity-quasi_uniform_surface
type-classified_item
type-area_measure
type-specified_item
type-certified_item
type-characterized_definition
type-transition_code
type-approved_item
```

Status: Dataset loaded and ready.

Attributes

```
config_control_design-application_context_requires_ap_definitio
```

What Else?

```
sizeof
query
where
wr1
not
and
config_control_design
```

What Not?

```
type
select
real
product_definition_formati
axis2_placement_3d
line
surface
b_spline_surface
curve
b_spline_curve
product_definition
definition
start_request
change_request
contract
```

Text

```
(* AIM long form for ISO 10303-203 amendment 1
ISO TC184/SC4/WG3 N916   Larry McKee
2000-05-04*)SCHEMA config_control_design;
CONSTANT  dummy_gri :
geometric_representation_item :=
representation_item('') ||
geometric_representation_item();  dummy_tri :
topological_representation_item :=
representation_item('')              ||
topological_representation_item();
END_CONSTANT;  TYPE ahead_or_behind =
ENUMERATION OF    (ahead,     behind);
END_TYPE; -- ahead_or_behind  TYPE
approved_item = SELECT
(product_definition_formation,
product_definition,     configuration_effectivity,
configuration_item,     security_classification,
change_request,     change,     start_request,
start_work,     certification,     contract);
END_TYPE; -- approved_item  TYPE area_measure
= REAL;  END_TYPE; -- area_measure  TYPE
axis2_placement = SELECT    (axis2_placement_2d,
   axis2_placement_3d);  END_TYPE; --
axis2_placement  TYPE b_spline_curve_form =
ENUMERATION OF    (polyline_form,     circular_arc,
   elliptic_arc,     parabolic_arc,     hyperbolic_arc,
unspecified);  END_TYPE; -- b_spline_curve_form
TYPE b_spline_surface_form = ENUMERATION OF
(plane_surf,     cylindrical_surf,     conical_surf,
spherical_surf,     toroidal_surf,
surf_of_revolution,     ruled_surf,
generalised_cone,     quadric_surf,
surf_of_linear_extrusion,     unspecified);
END_TYPE; -- b_spline_surface_form  TYPE
boolean_operand = SELECT    (solid_model);
END_TYPE; -- boolean_operand  TYPE certified_item
= SELECT    (supplied_part_relationship);
END_TYPE; -- certified_item  TYPE
change_request_item = SELECT
```

Figure 4: A sample HAMLET screen. To the left are the nearest concepts to your new design. The right side contains information from the chosen nearby design. This includes the original text as well as attributes pulled by the parser. In the nearest concepts, there exists certain concepts not found in the new design. These are shown in the *what else* list in the center. Also, the new design contains certain concepts not found in the nearest concepts. These are shown in the *what not* list (also, center).

results relevant to that user's current design task.

From the feedback loop, preference knowledge can be learned. Given a community of designers working on related tasks, HAMLET will be able to quickly learn what prior designs are relevant for that community.

2.3 Related Work

HAMLET draws on much of the literature on AI, information retrieval, text mining, and program comprehension. Formally, HAMLET is a *suggestion system* that augments standard queries with (a) suggestions that near to the current partially incomplete design and (b) suggestions of additions to the current design that would make it very unusual with respect to the space of all prior designs [11].

Having said that, the comprehension of archival technical documentation has certain attributes that make HAMLET's task different to other systems:

- Zhai et al. [27] discuss cross-collection mixture models that seek to discover latent common themes from large document collections. This work assumes that all documents are expressed in a simple "bag of words" model (i.e. no links between documents). In HAMLET, on the other hand, documents are stored in a connected graph showing neighborhoods of related terms.

- The HIPIKAT system of Čubranić and Murphy [24] explores comprehension of heterogeneous technical products. Software development projects produce a large number of artifacts, including source code, documentation, bug reports, e-mail, newsgroup articles, and version information. The information in these artifacts can be helpful to a software developer trying to perform a task, such as adding a new feature to a system. Unfortunately, it is often difficult for a software developer to locate the right information amongst the huge amount of data stored. HIPIKAT recommends relevant software development artifacts based on the context in which a developer requests help from HIPIKAT. While a landmark system, HIPIKAT is hard-wired into a particular set of development tools (ECLIPSE) and the scalability of the tool has not be demonstrated by the authors.

- Hill et al.'s DORA system [15] stores JAVA programs using a combination of "bag of words" as well as neighborhood hood information. Searching in DORA is two-fold process where:

  - Information retrieval on the bag of words finds candidate connections
  - Topology queries on the candidates' neighbors returns the strongly related terms.

  The drawback with DORA is that it assumes a homogeneous corpus (everything in DORA is a JAVA class) and its search algorithms will not scale to very large examples (since they are slower than linear time and memory).

3 Why HAMLET? (motivation)

3.1 Background

A recent NSF-funded workshop[1] highlighted current directions in long term technical document retention. While much progress was reported on:

- systems issues of handling and sharing very large data collection (e.g. SLASH)
- scalable methods of building customization views (e.g. iRODS),

there was little mention of the cognitive issues of how users might browse and synthesize data from massive data collections of technical documents.

For example, here at WVU, we are mid-way through a review of the use of STEP/EXPRESS for long term technical document retention[2]. STEP/EXPRESS is commonly used as an inter-lingua to transfer technical data between CAD/CAM packages. Strange to say, while STEP/EXPRESS is useful for transferring and understanding technical documents *today*, it does not appear to be suitable for understanding technical documents from *yesterday*.

In theory, there is nothing stopping STEP/EXPRESS from recording and storing all aspects of a project. In many ways, STEP/EXPRESS is as expressive as other technical document standards (e.g. UML). STEP/EXPRESS offers a generic method for storing part-of and isa information, constraints, types, and the rules associated with a technical document. However, in practice, the theoretical potential of STEP/EXPRESS is not realized for the following reasons.

3.1.1 Heterogeneity

The reality of archival systems is that STEP/EXPRESS documents are stored *along side* a much larger set of supporting documents in multiple formats. A recent study[3] concluded that

- 80 percent of business is conducted on unstructured information.
- 85 percent of all data stored is held in an unstructured format (e.g. the unstructured text descriptions of issues found in PITS).
- Unstructured data doubles every three months.

That is, if we can learn how to understand large heterogeneous collections that include STEP/EXPRESS knowledge as well as numerous other products in a wide variety of formats, it would be possible to reason and learn from a very wide range of data.

[1]Collaborative Expedition Workshop #74, June 10, 2008, at NSF. "Overcoming I/O Bottlenecks in Full Data Path Processing: Intelligent, Scalable Data Management from Data Ingest to Computation Enabling Access and Discovery". `http://colab.cim3.net/cgi-bin/wiki.pl?ExpeditionWorkshop/TowardScalableDataManagement_2008_06_10`

[2]See reports from Mucino.

[3]`http://www.b-eye-network.com/view/2098`

3.1.2 Incomplete meta-knowledge

A lot of work has focused on the creation of cached sets of EXPRESS schemas. Forty such *application protocols* (AP) have been defined [16] including AP-203 (for geometry) and AP-213 (for numerical control). The list of currently defined application protocols is very extensive (see Figure 5). These APs are the cornerstone of STEP tools: the tools offer specialized support and screen import/export facilities for the APs. While much effort went into their creation of these APs, very few have been stress-tested in the information systems field. That is, the majority of these APs have been *written* more than they have been *read* (exceptions: the above-mentioned AP-203 and AP-213 are frequently used and reused in 21^{st} century CAD/CAM manufacturing processes),

3.1.3 Incomplete tool support

Perhaps because of the relative immaturity of the APs, current CAD/CAM tools offer limited support for the STEP APs. While most tools support geometry (AP-203), the support for the other APs in Figure 5 is minimal (to say the least).

3.1.4 Incomplete design rationale support

From a cognitive perspective, STEP/EXPRESS does not support the entire design cycle. Rather, it only supports the last stages of design and not all of the interim steps along the way.

3.1.5 Limited Historical Use

For all the above reasons, highly structured technical documents in formats like STEP/EXPRESS are in the minority in the archival systems we have examined. We are aware of large STEP/EXPRESS repositories but these are often inaccessible for a variety of reasons.

While this situation might change in the future (e.g. if all the above issues were suddenly fixed and all organizations switch to using highly structured technical documentation), the historical record would still be starved for large numbers of examples.

3.2 Why Use HAMLET To Examine Data?

The most obvious reason for using HAMLET to look at a corpus of technical documents is its visualization value. Let's say that your collection contains thirty STEP documents and the EXPRESS schema that they all relate to. If you just looked through those documents, you'd be faced with a complete information overload. That's thirty-one separate text files to stare at, some with over a thousand lines of code.

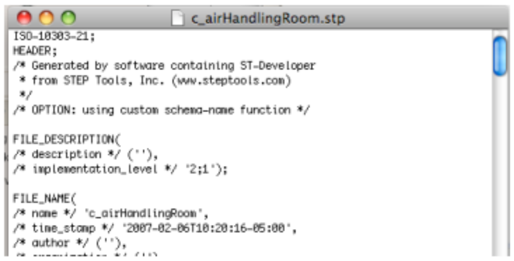
HAMLET provides an easy-to-use interface to visualize that data and make deductions based on the raw content (see Figure 6). The document panel lists only the documents that are relevant to your query (which can be a STEP design that you like). You can click on one of those documents ans view the raw text as well as key attributes of that file. This prevents the information overload associated with the raw files by only giving you relevant information in a more organized fashion.

AP	area
201	Explicit Drafting
202	Associative Drafting
203	Configuration Controlled Design
204	Mechanical Design Using Boundary Representation
205	Mechanical Design Using Surface Representation
206	Mechanical Design Using Wireframe Representation
207	Sheet Metal Dies and Blocks
208	Life Cycle Product Change Process
209	Design Through Analysis of Composite and Metallic Structures
210	Electronic Printed Circuit Assembly, Design and Manufacturing
211	Electronics Test Diagnostics and Remanufacture
212	Electrotechnical Plants
213	Numerical Control Process Plans for Machined Parts
214	Core Data for Automotive Mechanical Design Processes
215	Ship Arrangement
216	Ship Molded Forms
217	Ship Piping
218	Ship Structures
219	Dimensional Inspection Process Planning for CMMs
220	Printed Circuit Assembly Manufacturing Planning
221	Functional Data and Schematic Representation for Process Plans
222	Design Engineering to Manufacturing for Composite Structures
223	Exchange of Design and Manufacturing DPD for Composites
224	Mechanical Product Definition for Process Planning
225	Structural Building Elements Using Explicit Shape Rep
226	Shipbuilding Mechanical Systems
227	Plant Spatial Configuration
228	Building Services
229	Design and Manufacturing Information for Forged Parts
230	Building Structure frame steelwork
231	Process Engineering Data
232	Technical Data Packaging
233	Systems Engineering Data Representation
234	Ship Operational logs, records and messages
235	Materials Information for products
236	Furniture product and project
237	Computational Fluid Dynamics
238	Integrated CNC Machining
239	Product Life Cycle Support
240	Process Planning

Figure 5: STEP Application Protocols.

HAMLET's graph tools provide another key visualization advantage. The two-dimensional graph shows the nearby documents and how they relate to each other. For example, if your data is written in STEP, the graph maps how the STEP designs relate to their associated schema. HAMLET's three-dimensional graph provides a visual tool that allows you to quickly look at how close certain designs are to your query.

The power of HAMLET is not just in the formatting of data, it is also in the machine learning techniques that allow you to dive deeper into the data. Clustering algorithms and classifiers ensure that only relevant data is displayed, and user feedback mechanisms ensure that the experience is tailored to the individual user.

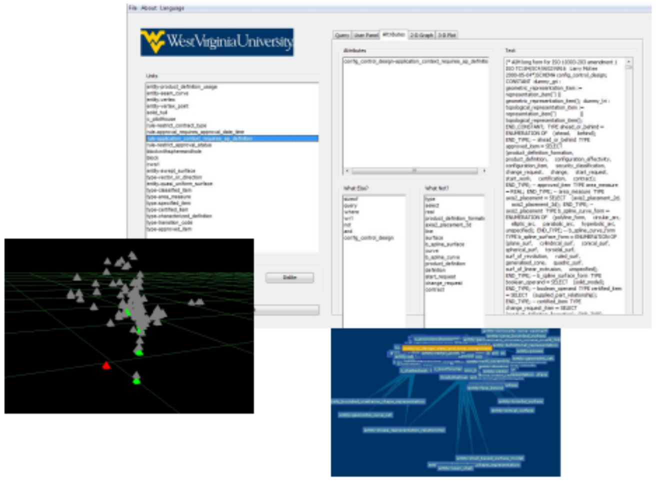


```
ISO-10303-21;
HEADER;
/* Generated by software containing ST-Developer
 * from STEP Tools, Inc. (www.steptools.com)
 */
/* OPTION: using custom schema-name function */

FILE_DESCRIPTION(
/* description */ (''),
/* implementation_level */ '2;1');

FILE_NAME(
/* name */ 'c_airHandlingRoom',
/* time_stamp */ '2007-02-06T10:20:16-05:00',
/* author */ (''),
```

Raw Text



Visualization with HAMLET

Figure 6: Looking at data using HAMLET has several visualization advantages

3.3 Under the Hood

3.3.1 Generic Parsing

Internally, HAMLET makes minimal assumptions about the form of the technical document:

- A document contains slots and slots can be atomic or point to other documents;.
- The network of pointers between documents presents the space of connected designs.

A generic parser class implements a standard access protocol for this internal model. By sub-classing that parser, it is possible to quickly process new documents types. Currently, HAMLET's parsers can import:

- STEP/EXPRESS
- Florida Law (XML)
- Text documents structured as follows: sub-headings within headings, paragraphs within sub-headings, sentences within paragraphs, words in sentences;
- JAVA: This JAVA import allows ready access to very large corpora of structured technical information (i.e. every open source JAVA program on the web). Hence, in the sequel, we will make extensive use of JAVA examples since that permits tests of scalability.

3.3.2 The Geometry of Design

HAMLET treats technical document comprehension as a geometric problem:

- Old designs are clustered into groups.
- A new design can be placed at some point around those clusters.
- To compute "what else," HAMLET finds the cluster nearest the new design and looks for differences between the new design and the average design in that cluster.
- To compute "what not," HAMLET looks for parts of the current design that are not usually found in the nearest cluster.

While simple in concept, the challenge of HAMLET is three-fold:

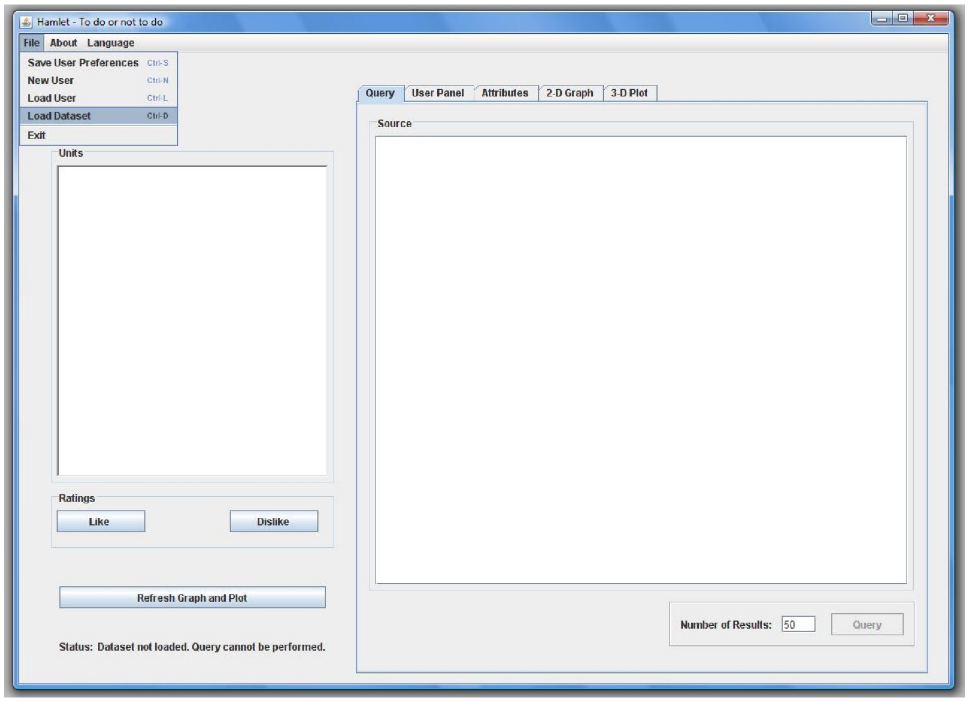
1. Doing all the above in a scalable manner; i.e. linear or sub-linear time processing. HAMLET handles this is a variety ways including methods borrowed from Google.
2. Doing all the above for a heterogeneous corpus. HAMLET handles multiple formats in the corpus by storing them all documents in a minimalistic internal format (a document contains slots and slots can be atomic or point to other documents).

3. While the minimal format permits rapid extension of HAMLET to other document types, it raises the issue of false alarms. Like any information retrieval task, HAMLET returns some false negatives (i.e. incorrect “what not” results) and false positives (i.e. incorrect “what else” results). HAMLET therefore builds profiles for each user based on their particular preferences.

4 A Session With HAMLET

4.1 The Preprocessor

HAMLET contains several components other than the UI shown below. One such component is the pre-processor, a tool used by HAMLET to generate datasets which can then be loaded into the UI allowing the user to query, rank, and visualize the documents found in the loaded dataset. The majority of all machine learning takes place within the pre-processor. This is where tasks like term frequency / document frequency generation, term selection, clustering, and learner training occurs. After being run through the pre-processor, each document within a collection is assigned a vector representation which describes what terms are present in the document and at what frequency.

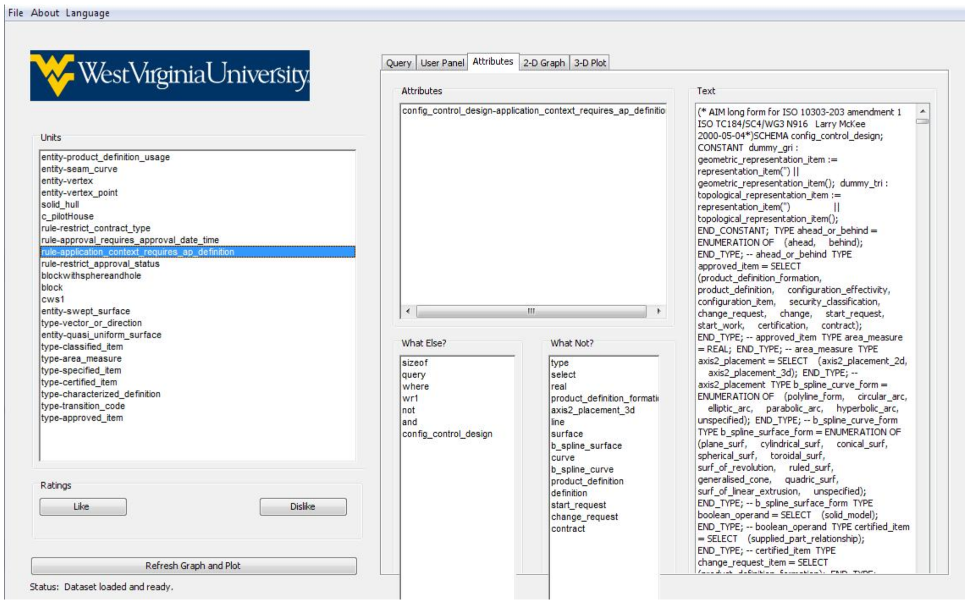


Status: Dataset not loaded. Query cannot be performed.

Figure 7: A HAMLET screen.

4.2 HAMLET output

Pictured below is HAMLET's output after running a given query (the unit data text box). The results are shown in the far left box. The far right boxes (what else and what not) describe the deltas between your query and the currently selected item in the list on the left side. *What else* can be described as "what do I need to add to my thing to make it more like the selected thing," while *what not* is more like "what do I need to remove from my thing to make it more like the selected thing." The sample dataset being run here was built from a corpus of STEP documents corrusponding to AP 203.



Units

```
entity-product_definition_usage
entity-seam_curve
entity-vertex
entity-vertex_point
solid_hull
c_pilotHouse
rule-restrict_contract_type
rule-approval_requires_approval_date_time
rule-application_context_requires_ap_definition
rule-restrict_approval_status
blockwithsphereandhole
block
cws1
entity-swept_surface
type-vector_or_direction
entity-quasi_uniform_surface
type-classified_item
type-area_measure
type-specified_item
type-certified_item
type-characterized_definition
type-transition_code
type-approved_item
```

Status: Dataset loaded and ready.

Attributes

```
config_control_design-application_context_requires_ap_definitio
```

What Else?

```
sizeof
query
where
wr1
not
and
config_control_design
```

What Not?

```
type
select
real
product_definition_formati
axis2_placement_3d
line
surface
b_spline_surface
curve
b_spline_curve
product_definition
definition
start_request
change_request
contract
```

Text

```
(* AIM long form for ISO 10303-203 amendment 1
ISO TC184/SC4/WG3 N916   Larry McKee
2000-05-04*)SCHEMA config_control_design;
CONSTANT  dummy_gri :
geometric_representation_item :=
representation_item('') ||
geometric_representation_item();  dummy_tri :
topological_representation_item :=
representation_item('')              ||
topological_representation_item();
END_CONSTANT;  TYPE ahead_or_behind =
ENUMERATION OF    (ahead,     behind);
END_TYPE; -- ahead_or_behind  TYPE
approved_item = SELECT
(product_definition_formation,
product_definition,     configuration_effectivity,
configuration_item,     security_classification,
change_request,     change,     start_request,
start_work,     certification,     contract);
END_TYPE; -- approved_item  TYPE area_measure
= REAL;  END_TYPE; -- area_measure  TYPE
axis2_placement = SELECT    (axis2_placement_2d,
   axis2_placement_3d);  END_TYPE; --
axis2_placement  TYPE b_spline_curve_form =
ENUMERATION OF    (polyline_form,     circular_arc,
   elliptic_arc,     parabolic_arc,     hyperbolic_arc,
unspecified);  END_TYPE; -- b_spline_curve_form
TYPE b_spline_surface_form = ENUMERATION OF
(plane_surf,     cylindrical_surf,     conical_surf,
spherical_surf,     toroidal_surf,
surf_of_revolution,     ruled_surf,
generalised_cone,     quadric_surf,
surf_of_linear_extrusion,     unspecified);
END_TYPE; -- b_spline_surface_form  TYPE
boolean_operand = SELECT    (solid_model);
END_TYPE; -- boolean_operand  TYPE certified_item
= SELECT    (supplied_part_relationship);
END_TYPE; -- certified_item  TYPE
change_request_item = SELECT
```

Figure 8: HAMLET output.

4.3 A Web of Connections

HAMLET is a framework that supports the both the *semantic* and the *syntactic* structure inherent in data. Displayed below is an example of the former, a web of hyper-links connecting relevant document to each other (generated from STEP/EXPRESS data). When this kind of information is combined with syntactic information (the actual text of a document, e.g. term counts) a powerful information retrieval system can be created that supports the ability to *walk* through the data. In our application, by clicking on a document in the graph, you are shown all of the documents connected to the selected document. By hopping from document to document and tweaking visualizations parameters along the way, it is possible to truly walk through the dataset.

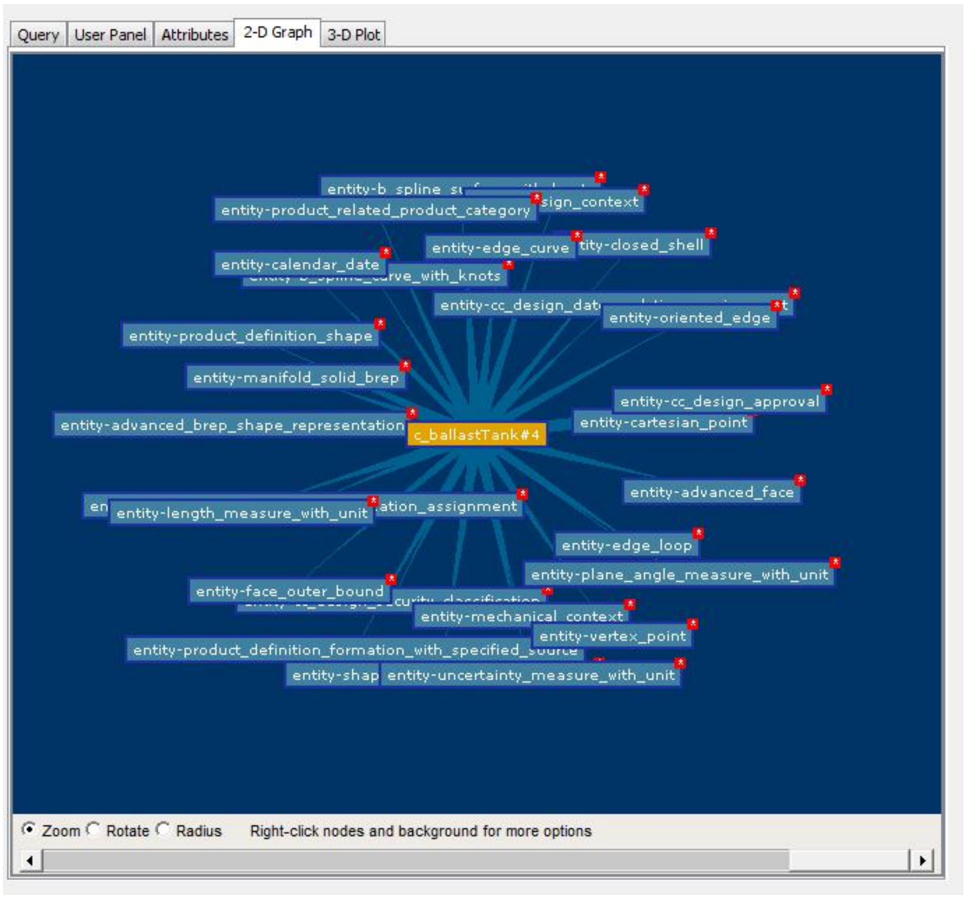

Query
User Panel
Attributes
2-D Graph
3-D Plot
entity-product_related_product_category
sign_context
entity-edge_curve
tity-closed_shell
entity-calendar_date
rve_with_knots
entity-cc_design_dat
entity-oriented_edge
entity-product_definition_shape
entity-manifold_solid_brep
entity-cc_design_approval
entity-advanced_brep_shape_representation
c_ballastTank#4
entity-cartesian_point
entity-advanced_face
entity-length_measure_with_unit
ation_assignment
entity-edge_loop
entity-plane_angle_measure_with_unit
entity-face_outer_bound
entity-mechanical_context
entity-vertex_point
entity-product_definition_formation_with_specified_source
entity-shap
entity-uncertainty_measure_with_unit
Zoom
Rotate
Radius
Right-click nodes and background for more options

Figure 9: Displaying connections.

4.4 3-D Visualization

In addition to visualizing 2-D semantic information, HAMLET also supports the ability to visualize each document vector as a point in 3-D space. To facilitate this, HAMLET utilizes dimensionality reduction in two stages. In the first stage, the list of all possible terms in a given collection is analyzed to determine the most relevant terms (this reduces the dimensionality from around 20,000 to 100). In the second stage, the 100 dimension document vectors are run through a fast (nearly linear) dimensionality reduction algorithm called FastMap which finds the intrinsic geometry in the 100 dimensional space and projects that into a 3-D space capable of visualization.

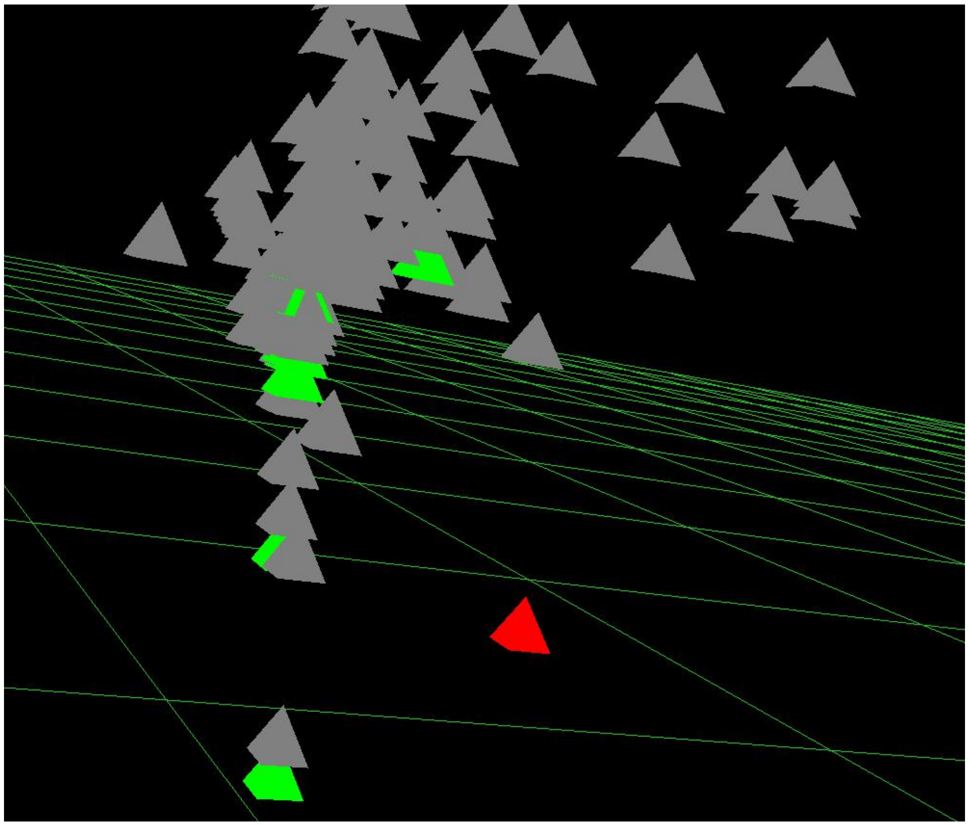


Figure 10: 3-D visualization. A new design (in red) floats near its nearest related concepts (in green). The gray points show parts of specifications that are less relevant to the new design. Note that this is a 2-D visualization of a 100-D space.

4.5 User Profiles

HAMLET gives each user a unique profile that is stored on the local machine. During a typical HAMLET session, the user can indicate whether they like or dislike a particular document. HAMLET utilizes the latest semantic analysis techniques to bring documents similar to those rated as liked to the top of query results, while filtering out the documents disliked. The weights applied to whole documents by the user ratings are also applied to the terms within that document, and the ”what-else/what-not” list is weighted by the sum of all ratings on each term in the list.

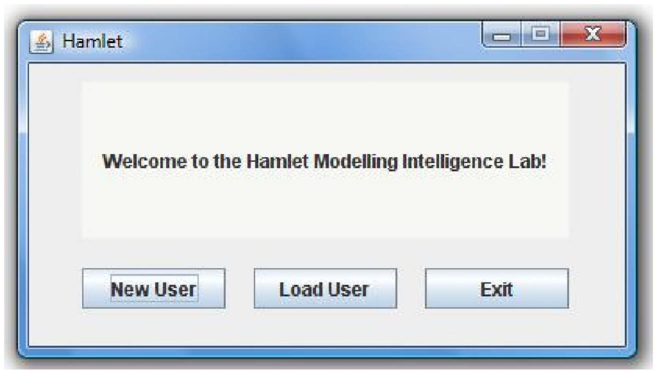


Figure 11: Logging in.

The user profile itself is stored in a comma-separated CSV file. Each row is a term and each column is a document (identified by its unique vector ID). Datasets are kept separate within the profile so that vector IDs do not overlap. A document that is liked receives a rating of 1, disliked are rated -1. A -2 means that the term is not found within that document. A total is kept at the end, which is used for ranking purposes.

```
%Timothy Menzies

%dataset #1
Term,Vector,Vector2,Vector3,Total
Bob,-1,1,1,1
Alice,-2,1,-2,1
Jimmy,-1,-2,1,0
```

Figure 12: A simple user profile.

5 How Does HAMLET Work? (the details)

This section offers a technical description of the internals of HAMLET. In that description, the term "document" will be used since that is consistent with the information retrieval literature. Note that HAMLET's "document" may be much smaller than, say, a Microsoft Word .doc file. A HAMLET "document" is just the smallest unit of comprehension for a particular type of entry in an archive. Indeed, depending on the parser being used, a "document" may be:

- An EXPRESS data type
- A JAVA method
- A paragraph in an English document
- Some XML snippet.
- All text associated with an archived engineering project

The methods described in this section are in a state of flux. HAMLET is a prototype system and we are constantly changing the internal algorithms. In particular:

- All slower-than-linear algorithms are being replaced with linear algorithms.
- Our preliminary experiments suggest that many common methods may in fact be superfluous for comprehension of technical documents. For example: (a) we may soon be dropping the stopping and stemming methods described below; (b) the value of discretization during InfoGain processing is not clear at this time.
- Any threshold value described in this section (e.g. using the top $k = 100$ Tf*IDF terms) will most probably change in the very near future as we tune HAMLET to problem of archival storage.

5.1 Parsing From Native Formats

HAMLET utilizes a generic parsing framework that provides an interface between existing parsed data and information retrieval, text mining and program comprehension methods supported by HAMLET. This allows both safe access to the data at run-time, as well as easy implementation. A brief, high- level overview of the main functions of this framework is discussed below.

5.1.1 Parsed Languages

Since HAMLET makes as few assumptions about a technical document as possible, any language could theoretically be parsed and used within the user interface. The HAMLET parsing API provides an interface for creating the XML format that the HAMLET interface reads. Parsers have been provided that process data authored in the following formats:

- STEP/Express
- Plain Text
- Florida Law (XML)
- HTML
- Java

The above formats were chosen according to:

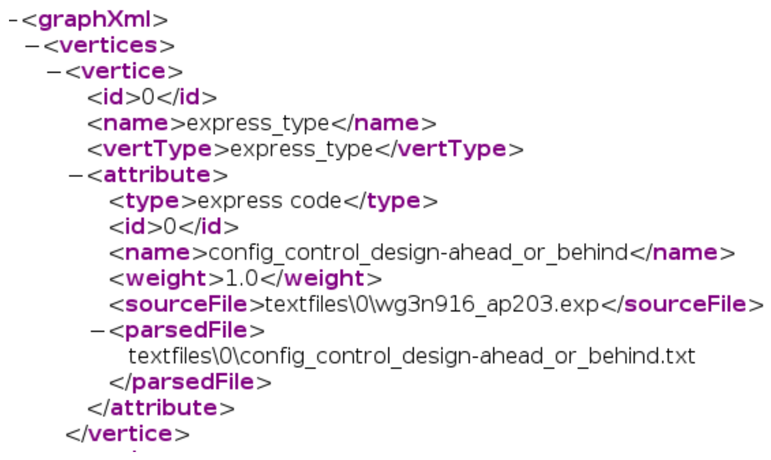
- Their relevance to this project- so STEP is highest;
- As well the availability of large corpora- so we use JAVA class libraries and a large XML dump pf 400 years of Florida real estate law.

5.1.2 HAMLET Parsing API

HAMLET's language-specific subparsers comb through individual files and pull out important bits of information (entities in STEP, methods in Java). While processing individual files, these subparsers collect information about each document. Some of that information includes pointers to the parsed information and information about what entities are used by others. The HAMLET generic parsing framework provides several methods to utilize these data attributes.

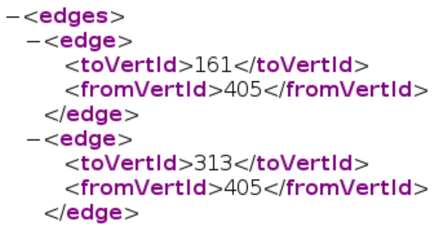
The most important function of the API is the generation of the GraphXML file. This file is the intermediary between the data set and HAMLET. It contains a list of each document (vertice) and the relationships between them. Other pertinent information, such as file pointers and document statistics, is stored in the form of attributes for each document vertice. From a higher level, the collection of these pointers to files gives a view of the region of interconnected designs, giving HAMLET the ability to make its decisions and provide suggestions based on what it has already learned.

For certain language imports, such as Java or STEP, HAMLET utilizes edge generation to determine the relationship of one design to another. For instance, if a call graph is generated on a set of Java source files, an edge can be placed between a multitude of methods and calls made to and from them. The XML graph generated by the parsing API includes both the document vertices and the edges that connect them. This is essential for visualization purposes and provides a wealth of syntactical information.



```
-<graphXml>
  -<vertices>
    -<vertice>
        <id>0</id>
        <name>express_type</name>
        <vertType>express_type</vertType>
      -<attribute>
          <type>express code</type>
          <id>0</id>
          <name>config_control_design-ahead_or_behind</name>
          <weight>1.0</weight>
          <sourceFile>textfiles\0\wg3n916_ap203.exp</sourceFile>
        -<parsedFile>
            textfiles\0\config_control_design-ahead_or_behind.txt
          </parsedFile>
        </attribute>
      </vertice>
```

Figure 13: One vertice and the information associated with it.



```
-<edges>
  -<edge>
      <toVertId>161</toVertId>
      <fromVertId>405</fromVertId>
    </edge>
  -<edge>
      <toVertId>313</toVertId>
      <fromVertId>405</fromVertId>
    </edge>
```

Figure 14: Edges in the graph XML file.

5.2 TF*IDF

In order to perform mathematical operations and algorithms on documents and the text that they contain, we must first transform them into a representative mathematical object. The standard representation of a document is a vector in the space of all available terms. For example, the phrase:

$$\begin{array}{c} The\ quick\ brown\ dog\ was\ very \\ quick,\ very\ brown,\ and\ very\ dog\ like. \end{array} \tag{1}$$

Will be turned into a vector which looks something like this:

$$Phrase = [1\ 2\ 2\ 2\ 1\ 3\ 1\ 1] \tag{2}$$

with each index of the above vector corresponding the a dimension which comes from the term list (in this case, the dimensions are the, quick, brown, dog, was, very, like)

Tf*Idf is shorthand for "term frequency times inverse document frequency." This calculation models the intuition that jargon usually contains technical words that appear a lot, but only in a small number of paragraphs. For example, in a document describing a space craft, the terminology relating to the power supply may appear frequently in the sections relating to power, but nowhere else in the document.

Calculating Tf*Idf is a relatively simple matter:

- Let there be $Words$ number of documents;
- Let some word I appear $Word[I]$ number of times inside a set of $Documents$;
- Let $Document[I]$ be the documents containing I.

Then:

$$Tf * Id = Word[i]/Words * log(Documents/Document[i])$$

The standard way to use this measure is to cull all but the k top Tf*Idf ranked stopped, stemmed tokens. This study used $k = 100$.

5.3 Dimensionality Reduction

A major issue within HAMLET is *dimensionality reduction*. Standard AI learning methods work well for problems that are nearly all fully described using dozens (or fewer) attributes [25]. But a corpus of archival technical documents must process thousands of unique words, and any particular document may only mention a few of them [1, 23]. Therefore, before we can apply learning to technical document comprehension, we have to reduce the number of dimensions (i.e. attributes) in the problem.

There are several standard methods for dimensionality reduction such as *tokenization, stop lists, stemming, Tf*IDF*, *InfoGain*, PCA, and FastMap. All these methods are discussed below.

5.3.1 Tokenization

In HAMLET's parser, words are reduced to simple tokens via (e.g.) removing all punctuation remarks, then sending all upper case to lower.

5.3.2 Stop lists

Another way to reduce dimensionality is to remove "dull" words via a *stop list* of "dull" words. Figure 15 shows a sample of the stop list used in HAMLET. Figure 15 shows code for a stop-list function.

a	about	across	again	against
almost	alone	along	already	also
although	always	am	among	amongst
amongst	amount	an	and	another
any	anyhow	anyone	anything	anyway
anywhere	are	around	as	at
...	...	...	...	...

Figure 15: 24 of the 262 stop words used in this study.

5.3.3 Stemming

Terms with a common stem will usually have similar meanings. For example, all these words relate to the same concept.

- CONNECT
- CONNECTED
- CONNECTING
- CONNECTION
- CONNECTIONS

Porter's stemming algorithm [21] is the standard stemming tool. It repeatedly replies a set of pruning rules to the end of words until the surviving words are unchanged. The pruning rules ignore the semantics of a word and just perform syntactic pruning (e.g. Figure 16).

RULE			EXAMPLE		
ATIONAL	->	ATE	relational	->	relate
TIONAL	->	TION	conditional	->	condition
			rational	->	ration
ENCY	->	ENCE	valency	->	valence
ANCY	->	ANCE	hesitancy	->	hesitance
IZER	->	IZE	digitizer	->	digitize
ABLY	->	ABLE	conformably	->	conformable
ALLY	->	AL	radically	->	radical
ENTLY	->	ENT	differently	->	different
ELY	->	E	vilely	->	vile
OUSLY	->	OUS	analogously	->	analogous
IZATION	->	IZE	vietnamization	->	vietnamize
ATION	->	ATE	predication	->	predicate
ATOR	->	ATE	operator	->	operate
ALISM	->	AL	feudalism	->	feudal
IVENESS	->	IVE	decisiveness	->	decisive
FULNESS	->	FUL	hopefulness	->	hopeful
OUSNESS	->	OUS	callousness	->	callous
ALITY	->	AL	formality	->	formal
IVITY	->	IVE	sensitivity	->	sensitive
BILITY	->	BLE	sensibility	->	sensible

Figure 16: Some stemming rules.

Porter's stemming algorithm has been coded in any number of languages[4] such as the Perl *stemming.pl* used in this study.

5.3.4 InfoGain

According to the $InfoGain$ measure, the *best* words are those that *most simplifies* the target concept (in our case, the distribution of frequencies seen in the terms). Concept "simplicity" is measured using information theory. Suppose a data set has 80% severity=5 issues and 20% severity=1 issues. Then that data set has a term distribution C_0 with terms $c(1) = cat$, $c(2) = dog$ etc with frequencies (say) $n(1) = 0.8$, $n(2) = 0.2$ etc then number of bits required to encode that distribution C_0 is $H(C_0)$ defined as follows:

$$\left.\begin{array}{rcl} N &=& \sum_{c \in C} n(c) \\ p(c) &=& n(c)/N \\ H(C) &=& -\sum_{c \in C} p(c) log_2 p(c) \end{array}\right\} \quad (3)$$

After discretizing numeric data[5] then if A is a set of attributes, the number of bits required to encode a class after observing an attribute is:

$$H(C|A) = -\sum_{a \in A} p(a) \sum_{c \in C} p(c|a) log_2(p(c|a)$$

[4]`http://www.tartarus.org/martin/PorterStemmer`

[5]E.g. given an attribute's minimum and maximum values, replace a particular value n with $(n - min)/((max - min)/10)$. For more on discretization, see [9].

The highest ranked attribute A_i is the one with the largest *information gain*; i.e the one that most reduces the encoding required for the data *after* using that attribute; i.e.

$$InfoGain(A_i) = H(C) - H(C|A_i) \tag{4}$$

where $H(C)$ comes from Equation 3. In this study, we will use InfoGain to find the top $N = 10$ most informative tokens.

5.3.5 TF*IDF Ranking

A similar way of reducing the number of terms is by using the summation of the TF*IDF scores for each term. This method gives each a term a ranking tr_i each term using the following equation:

$$tr_i = \sum_{d \in D} TfIdf_{i,d} \tag{5}$$

where D is the entire set of documents and $TfIdf_{i,d}$ is the Tf*Idf value for term i and document d. After this has been computed for each term, a simple sort over all terms will give us the most important terms, as defined by their Tf*Idf scores. See Figure 17 for a list of real terms returned form a STEP dataset.

The benefit of using this method over InfoGain is not having to discretize the Tf*Idf values. By using the Tf*Idf values as they are, we can bypass an extra computational step. Additionally, this approach doesn't have to compute a new metric as InfoGain does, it simply sums the existing Tf*Idf scores which is computationally faster than computing InfoGain.

`cartesian_point`	`type`	`oriented_edge`	`subtype`	`entity`
`label`	`sizeof`	`self`	`query`	`select`
`name`	`direction`	`edge_curve`	`where`	`wr1`
`text`	`real`	`set`	`not`	`description`
`for`	`rule`	`items`	`typeof`	`supertype`
`...`	`...`	`...`	`...`	`...`

Figure 17: 30 of the 100 key terms found in a STEP dataset using TF*IDF ranking

5.3.6 PCA and FastMap

Numerous data mining methods check if the available features can be combined in useful ways. These methods offer two useful services:

1. Latent important structures within a data set can be discovered.
2. A large set of features can be mapped to a smaller set, then it becomes possible for users to manually browse complex data.

For example, principal components analysis (PCA) [7] has been widely applied to resolve problems with structural code measurements; e.g. [20]. PCA identifies the distinct orthogonal sources of variation in a data sets, while mapping the raw features onto a set of uncorrelated features that represent essentially the same information contained in the original data. For example, the data shown in two dimensions of Figure 18 (left-hand-side) could be approximated in a single latent feature (right-hand-side).

Since PCA combines many features into fewer latent features, the structure of PCA-based models may be very simple. For example, previously [3], we have used PCA and a decision tree learner to find the following predictor for defective software modules:

```
if   $domain_1 \leq 0.180$
then NoDefects
else if $domain_1 > 0.180$
     then    if $domain_1 \leq 0.371$ then NoDefects
     else    if $domain_1 > 0.371$ then Defects
```

Here, "$domain_1$" is one of the latent features found by PCA. This tree seems very simple, yet is very hard to explain to business clients users since "$domain_1$" is calculated using a very complex weighted sum (in this sum, $v(g), ev(g), iv(g)$ are McCabe or Halstead static code metrics [18, 13] or variants on line counts):

$$\begin{aligned} domain_1 = {} & 0.241 * loc + 0.236 * v(g) \\ & +0.222 * ev(g) + 0.236 * iv(g) + 0.241 * n \\ & +0.238 * v - 0.086 * l + 0.199 * d \\ & +0.216 * i + 0.225 * e + 0.236 * b + 0.221 * t \\ & +0.241 * lOCode + 0.179 * lOComment \\ & +0.221 * lOBlank + 0.158 * lOCodeAndComment \\ & +0.163 * uniq_Op + 0.234 * uniq_Opnd \\ & +0.241 * total_Op + 0.241 * total_Opnd \\ & +0.236 * branchCount \end{aligned} \tag{6}$$

Nevertheless, such latent dimensions can be used to generate visualizations that show users spatial distances between concepts in technical documents. For example, Figure 19 shows a 100-D space of prior designs converted to a 3-D representation. In the conversion process, the three top-most domains were computed and the 100-D space mapped to the 3-D space.

PCA is the traditional method of performing dimensionality reduction. It suffers from scale-up problems (for large data sets with many terms, the calculation of the correlation matrix between all terms is prohibitively computationally expensive). FastMap is a heuristic stochastic algorithm that performs the same task as PCA, but do so in far less time and memory [10]. Our own experiments with the two methods showed that both yield similar structures.

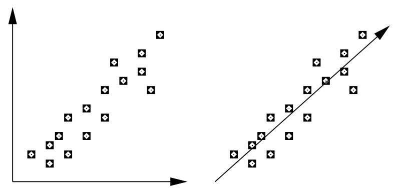


Figure 18: The two features in the left plot can be transferred to the right plot via one latent feature.

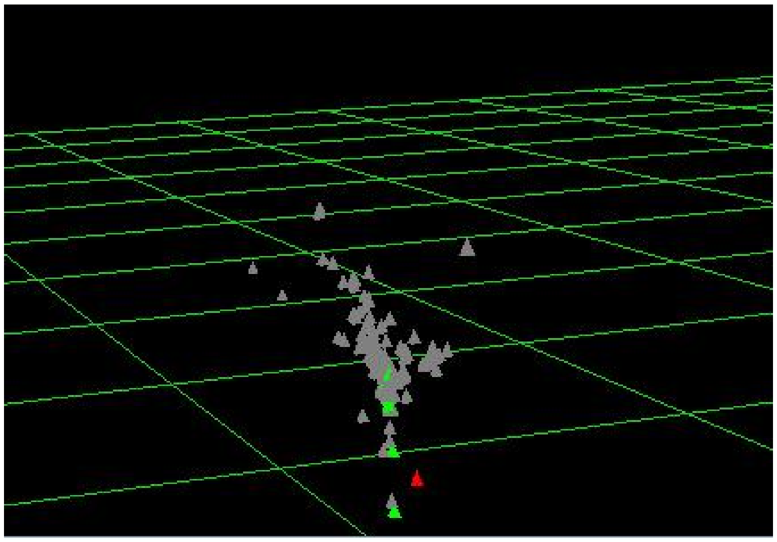


Figure 19: A 100-D space presented in 3-D. The point in red shows the current design and the four points in green show the nearest 4 technical documents (and the gray points show other documents that have been “trimmed” away using the techniques discussed below).

5.4 Clustering

HAMLET adopts a geometric view of technical and other textual documents floating in an N-dimensional space (those dimensions maybe have also been reduced by the above methods). The documents are clustered and new designs can be critiqued with respect to their position relative to old designs. The process of clustering allows similar designs (or textual documents) to be placed into groups based upon some defined similarity measure in this N-dimensional space. This is equivalent to asking, ”What *types* of things are out there?”

5.4.1 K-Means

K-Means is a clustering algorithm that, when given a dataset of unidentified objects, it will group those items into k groups based on some given similarity measure. The algorithm is described in Figure 20. For an example of the algorithm in operation, see Figure 21.

While k-means may be sufficiently accurate, there are significant drawbacks. Most notably is the speed (or lack thereof). Due to the k-means algorithm having to compute distances from every item to every cluster. In situations where the cosine similarity distance measure is used, computing the distance between points can be an expensive operation (this is another place dimensionality reduction helps out). In recent tests comparing clustering algorithm run-times, k-means was found to be up to 500 times

```
i=0
Partitioning the input points into k initial sets, either at random or using some heuristic data.
do
    Calculates the mean point, or centroid, of each set or cluster.
    Constructs a new partition, by associating each point with the closest centroid.
    Recalculate the centroids for the newly partitioned cluster
    i = i + 1
while($i \leq maxIterations$ or no point changes set membership)
```

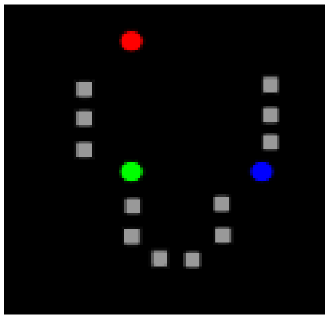
Figure 20: K-Means algorithm

slower than another algorithm, GENIC, which we discuss further down.

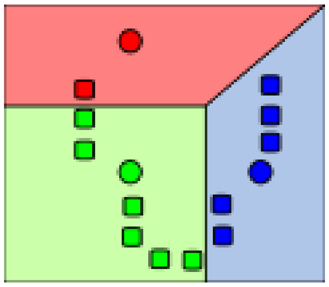
Another problem with k-means is determining what value of k should be used. Note the usability issues with requiring a user to pre-specify k: isn't this the kind of tedious detail that the computer should be telling us?

There are many techniques for automatically discovering an approximate value of k, all of which include several rounds of initial guesses, trying various values around the guess, then returning the k value that yields the best classification results. The problem with these techniques is that K-Means is a slow algorithm- requiring it to run many times is impractical for large corpora.

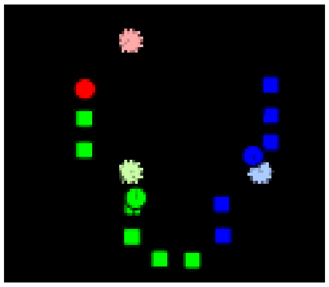
Step1: Here, we show some initial data points and the centroids generated based on random assignment



Step2: Points are associated with the nearest centroid:



Step3: Next, we recompute centroid using new associations and update the stored centroid:



Steps 2 & 3 are repeated until one of the two convergences criteria are reached.

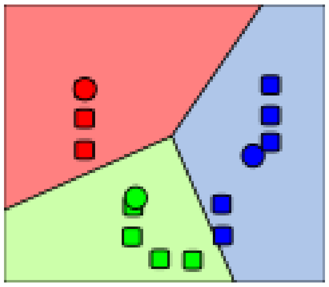


Figure 21: Example of K-means

5.4.2 Canopy Clustering

A naive clustering algorithm runs in $O(N^2)$ where N is the number of terms being clustered and all terms are assessed with respect to all other terms. For large archival collections, this is too slow. Various improvements over this naive strategy include ball trees, KD-trees and cover trees [2]. While all these methods are useful, their ability to scale to very large examples is an open question.

An alternative to traditional clustering methods is *canopy clustering* [5, 19]. It is intended to speed up clustering operations on large data sets, where using another algorithm directly may be impractical because of the size of the data set. In a standard clustering algorithms, two items are compared to determine some measure of how similar or different they are. There are several distance measures used for different domains (euclidean, cosine, manhattan, etc.), the draw back to all of these is that they are all relatively computationally expensive. The secret to canopy clustering's greater performance over conventional clustering techniques is it's use of two distance measures, one being approximately accurate but computationally cheap and the other being more accurate, however more expensive. To take advantage of the cheap distance metric, two passes are taken over the dataset. In the first pass, the cheap distance measure is used to determine *canopies*, which are groups of approximately close things. In the second pass, the more expensive distance measure is used. If any two items being compared do not share a canopy, then their distance is assumed to be infinite and no further comparison is done. By doing this, canopy clustering prevents having to perform n^2 comparisons at each step through the clustering algorithm.

The algorithm proceeds as follows:

- Cheaply partition the data into overlapping subsets, called 'canopies' (see Figure 22);
- Perform more expensive clustering, but only between these canopies.

In the case of text mining applications like HAMLET, the initial cheap clustering method can be performed using an inverted index; i.e. a sparse matrix representation in which, for each word, we can directly access the list of documents containing that word. The great majority of the documents, which have no words in common with the partial design constructed by the engineering, need never be considered. Thus we can use an inverted index to efficiently calculate a distance metric that is based on (say) the number of words two documents have in common.

5.4.3 GENIC

GENIC is a generalized incremental clustering algorithm developed by Gupta and Grossman [12] that provides HAMLET with two useful services:

- Scalability: Since GENIC was designed with streaming data in mind, it only has a single pass through the data to work with. Because of this, it scales linearly, which is a requirement if HAMLET is to scale to large corpora.
- An likely estimate for k: By using stochastic methods, GENIC can be given an initial k equal to the number of items (each item is its own clusters) and

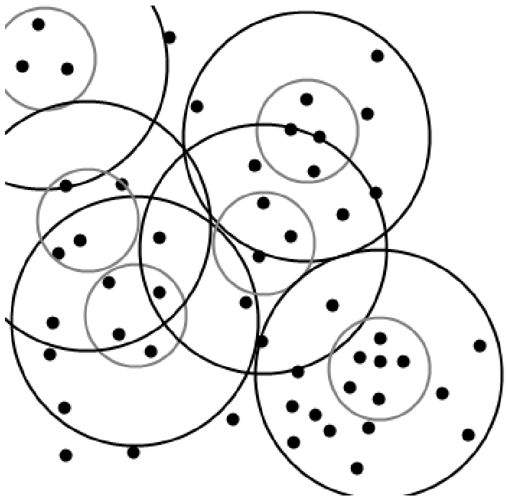


Figure 22: The darker circle represents all points in a given canopy, points in the smaller circles cannot be used as a new canopy center.

prune away unlikely clusters with each generation, giving a realistically estimated value for k after the last generation.

Here is how GENIC works:

1. **Select parameters**
   - Fix the number of centers k.
   - Fix the number of initial points m.
   - Fix the size of a generation n.
2. **Initialize**
   - Select m points, $c_1, ..., c_m$ to be the initial candidate centers.
   - Assign a weight of w_i = 1 to each of these candidate centers.
3. **Incremental Clustering** For each subsequent data point p in the stream: do
   - $Count$ = $Count$ + 1
   - Find the nearest candidate center c_i to the point p
   - Move the nearest candidate center using the formula

$$c_i = \frac{(w_i * c_i + p}{w_i + 1} \tag{7}$$

- Increment the corresponding weight

$$w_i = w_i + 1 \tag{8}$$

- When $Count\ mod\ n$ = 0, goto Step 4

4. **Generational Update of Candidate Centers**
   When $Count$ equals $n, 2n, 3n$, ..., for every
   center c_i in the list L of centers, do:

   - Calculate its probability of survival using the formula

   $$p_i = \frac{w_i}{\sum_{i=1}^{n} w_i} \tag{9}$$

   - Select a random number δ uniformly from [0,1]. If p_i ¿ δ, retain the center c_i in the list L of centers and use it in the next generation to replace it as a center in the list L of centers.
   - Set the weight w_i = 1 back to one. Although some of the points in the stream will be implicitly assigned to other centers now, we do not use this information to update any of the other existing weights.
   - Goto step 3 and continue processing the input stream

5. **Calculate Final Clusters** The list L contains the m centers. These m centers can be grouped into the final k centers based on their Euclidean distances.

GENIC has been shown in the literature to have a clustering error of less 1% [12]. Further, when tested on STEP schemas, it runs 100 to 1000s of times faster than k-means, as shown in Figure 23 and Figure 24.

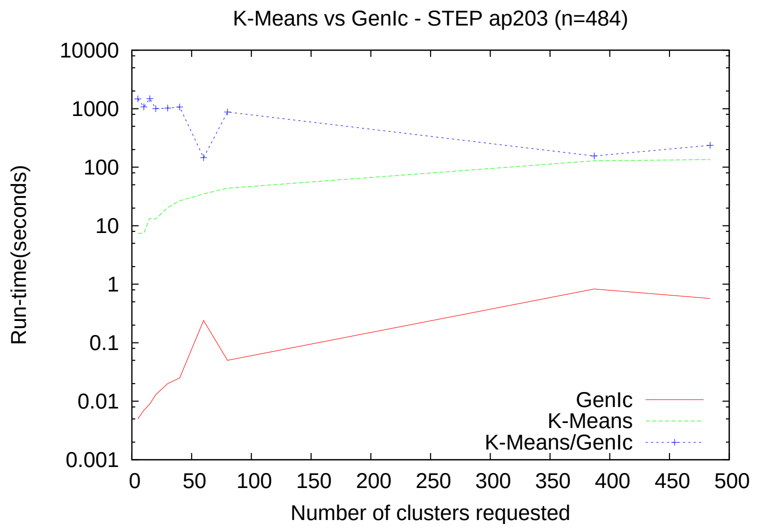


Figure 23: Clustering results of STEP AP203 (configuration controlled design).

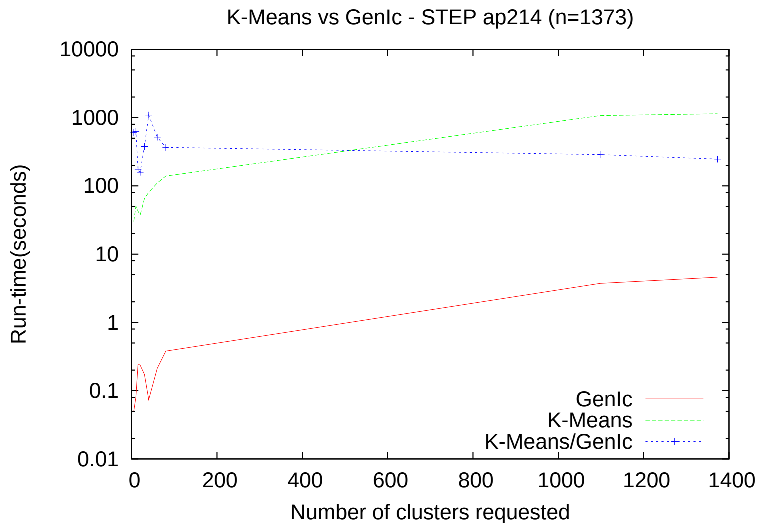


Figure 24: Clustering results of STEP AP214 (core data for automotive mechanical design processes).

5.5 Classification Within HAMLET

A key task with HAMLET is recognizing which cluster is nearest the partial design offered by HAMLET's user. The challenge is doing this in both a quick and effective way. However, as often is the case in problems of computer science, speed and accuracy are trade-offs.

5.5.1 Naive Bayes

Bayesian classifiers, and more generally the Naive Bayes algorithm, are simple statistical learning schemes. They have seen a lot of use in the Machine Learning field because they are fast, use very little memory, and are trivial to implement.

Naive Bayes is an application of Bayes' Theorem, relating the probability of event H given evidence E_i, a prior probability for a class $P(H)$, and a posteriori probability $P(H|E)$:

$$P(H|E) = \prod_i P(E_i|H)\frac{P(H)}{P(E)} \tag{10}$$

The classification with the highest probability is returned. The above assumes discrete attributes. To deal with numeric values, a features mean μ and standard deviation σ are used in a Gaussian probability function [26]:

$$f(x) = 1/(\sqrt{2\pi}\sigma)e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

These classifiers are called ”naive” because they assume that all attributes are equally important and statistically independent. Although these assumptions are almost never correct, Domingos and Pazzini have shown that the independence assumption is a problem in a vanishingly small number of cases [8]. On average, Naive Bayes classifiers perform as well, if not better than, more complex classification algorithms.

5.5.2 TWCNB

Rennie et al. [22] report a variant of a Naive Bayes classifier called Transformed Weight-normalized Complement Naive Bayes (TWCNB) that uses normalized Tf*IDF counts with the following properties:

- It handles low frequency data sets;
- It performs almost as well as more complex support vector machine implementations;
- Better yet, it is very simple to implement and runs in linear time (which makes it suitable for scaling up to a very large corpus).

By using an optimization on the Naive Bayes classifier called the TWCNB we can achieve near-state-of-the-art accuracy with state-of-the-art speed. This variant of Naive Bayes is highly optimized for text classification by doing the following:

- Transforming inherently non-gaussian text distributions (power law) into a guassian to fit with Naive Bayes's normal assumption
- Normalizes tfidf values to ensure the learner does not favor large or small documents

- Reverses the standard Naive Bayes likelihood function so instead of looking for things that are like a given target, we look for things are *not* like the given target, by doing this, TWCNB is able to avoid clusters with relatively low counts

Because of the nature of the data we are dealing with, a key requirement for our classifier is that it handles low frequency counts: we anticipate that archival data sets will contain many terms, but only very few of them will appear in any particular document or the user's partial design. This makes TWCNB an excellent candidate for HAMLET. We are currently experimenting with our own implementation of TWCNB.

5.6 Trimming

Trimming is a simple heuristic to prune remote points. It is a fast and simple method for focusing design reviews on near-by concepts.

Trimming runs like this:

- The user specifies some max distance measure N.
- The N nearest documents to the user's partially specified design are accessed. These documents are sorted $1, 2, 3, 4...N$ according to their distance to the current design.
- The delta between document i and $i + 1$ in the sort order is computed and the document i with the maximum delta (most difference between it and $i + 1$) is declared "the trim point".
- All documents $i + 1, i + 2, ...N$ are trimmed away.

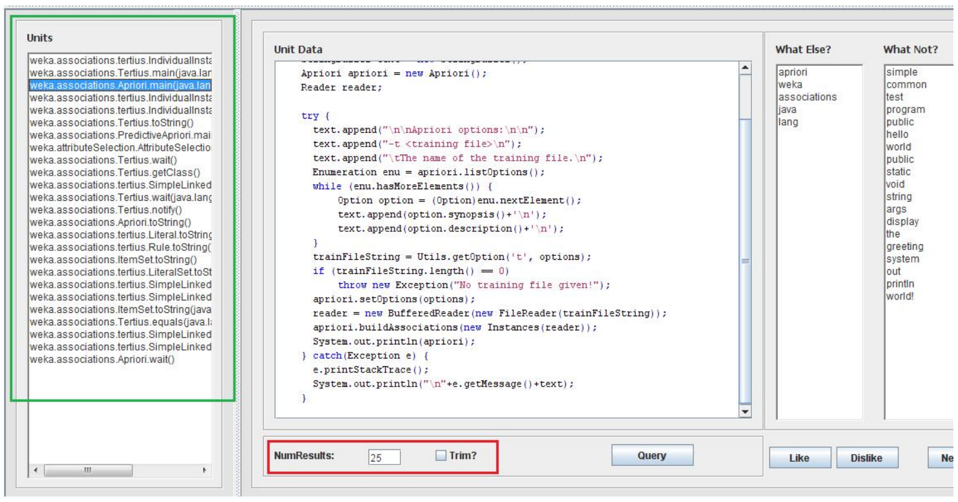
For example, Figure 25 show the number of related documents before and after trimming to a maximum depth of $N = 25$.

5.7 Query Results and What Else/What Not

After classifying the user's design using the above mentioned bayesian method, we can drastically reduce the search space of potentially similar designs. This is possible because after finding the most likely cluster (i.e. type of design), we only consider items within that cluster when finding designs similar to the user's query. After all potentially similar designs have been identified, k-NN (k-Nearest Neighbor where k is part of the query) is used to determine the most relevant designs within the set of potentials.

After returning the ranked potentially similar items, the user is then given the option of exploring the differences between the query (their design) and the results (designs indexed in HAMLET). Comparisons can be done between the query and either an individual item in the result list (ex. Product XYZ) or an item representative of all items in the same cluster (type) (or at least the mathematical representation of one, as this item may not be tangible). The delta generated by either of these comparisons comprises two lists:

- What to add (the "what else" list).



```
weka.associations.tertius.IndividualInsta
weka.associations.Tertius.main(java.lan
weka.associations.Apriori.main(java.lan
weka.associations.tertius.IndividualInsta
weka.associations.tertius.IndividualInsta
weka.associations.Tertius.toString()
weka.associations.PredictiveApriori.mai
weka.attributeSelection.AttributeSelectio
weka.associations.Tertius.wait()
weka.associations.Tertius.getClass()
weka.associations.tertius.SimpleLinked
weka.associations.Tertius.wait(java.lang
weka.associations.Tertius.notify()
weka.associations.Apriori.toString()
weka.associations.tertius.Literal.toString
weka.associations.tertius.Rule.toString(
weka.associations.ItemSet.toString()
weka.associations.tertius.LiteralSet.toSt
weka.associations.tertius.SimpleLinked
weka.associations.tertius.SimpleLinked
weka.associations.ItemSet.toString(java
weka.associations.Tertius.equals(java.la
weka.associations.tertius.SimpleLinked
weka.associations.tertius.SimpleLinked
weka.associations.Apriori.wait()
```

```java
    Apriori apriori = new Apriori();
    Reader reader;

    try {
      text.append("\n\nApriori options:\n\n");
      text.append("-t <training file>\n");
      text.append("\tThe name of the training file.\n");
      Enumeration enu = apriori.listOptions();
      while (enu.hasMoreElements()) {
          Option option = (Option)enu.nextElement();
          text.append(option.synopsis()+'\n');
          text.append(option.description()+'\n');
      }
      trainFileString = Utils.getOption('t', options);
      if (trainFileString.length() == 0)
          throw new Exception("No training file given!");
      apriori.setOptions(options);
      reader = new BufferedReader(new FileReader(trainFileString));
      apriori.buildAssociations(new Instances(reader));
      System.out.println(apriori);
    } catch(Exception e) {
      e.printStackTrace();
      System.out.println("\n"+e.getMessage()+text);
    }
```

What Else?	What Not?
apriori	simple
weka	common
associations	test
java	program
lang	public
	hello
	world
	public
	static
	void
	string
	args
	display
	the
	greeting
	system
	out
	println
	world!

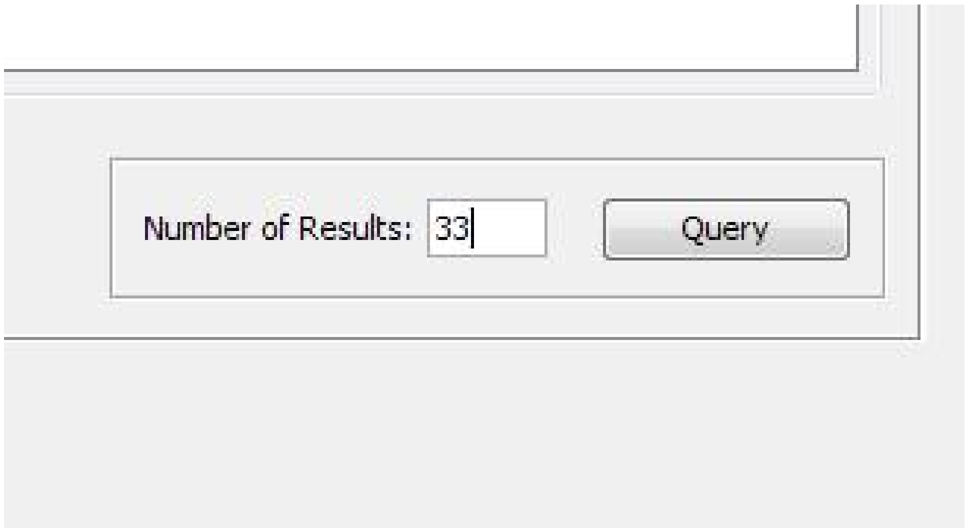


Figure 25: Trimming is a simple heuristic to prune remote points. It is a fast and simple method for focusing design reviews on near-by concepts. The top picture, left-hand-side, shows in green the 25 documents closest to some partial design developed by the user. The contents of the third closest document, highlighted in blue, is shown in the center screen. The bottom picture shows the same set of nearby documents, with trimming enabled (see the check box shown in red). Note that now only four documents are displayed to the user.

- What to remove (the “what not” list).

Given the treatment as designs as bags (documents) containing stuff (words), an algorithm to generate what else/what not based in set theory was the easiest to conceive. If each design is a set of words that appear somewhere in its documentation, then the formula’s for what else and what not are as follows:

- $$WhatElse(D, T) = T - D \tag{11}$$

- $$WhatNot(D, T) = D - T \tag{12}$$

where T is a target design or document (bag of words) and D is the design or document currently being evaluated.

$$P(H|E) = \prod_i P(E_i|H) \frac{P(H)}{P(E)} \tag{13}$$

5.8 User Profiling with the Rochio Algorithm

All information retrieval systems, including HAMLET, suffer from false alarms; i.e. returning query results that the user does not find relevant.

The Rochio algorithm is a standard method for pruning the results of an informational retrieval search with user feedback [14, 6] The algorithm reports the delta between the positive document vectors (that related to membership of the positive examples approved by the user) and the negative ones (that relate to membership of the negative examples disapproved by the user).

Given a set of documents D_q encompassed by query Q, you can separate the documents into two distinct subsets of liked (L_q) and disliked (U_q) documents. The normalized TFIDF vectors of those two subsets are summed and weighted by tuning parameters (α,β, and χ) that are determined via experimentation but Joachims recommends weighting the positive information four times higher than the negative [17]. They are then divided by the size of each set. These summations are then used to tune the original query vector as follows:

$$Q_n = \alpha Q + \frac{\beta}{|L_q|} \sum_{d \in L_q} d - \frac{\chi}{|U_q|} \sum_{d \in U_q} d \tag{14}$$

According to Dekhtyar et al /citehayes07, placing emphasis on the positive elements (liked documents in our example) may improve the recall (new relevant articles may be found) while emphasizing the negative (disliked documents) may affect precision (false positive may be removed).

We are currently exploring methods to augment Rochio with TWCNB.

6 Next Steps

Plan to throw one away; you will anyhow.
- "The Mythical Man-month" (1975)

HAMLET is a tool suitable for in-house use in a research lab. Much work is required before it can be released as a stand-alone tool for end-users, see Figure 26.

Note: The road-map in Figure 26 has changed little from last time (Aug 30):

1. As to point 1, we have yet to hit the JAVA wall that makes us recode in "C" (the GENIC experience suggests that smart algorithms can take us further than changing the implementation language).
2. We have some promising results regarding point 2 (scalability: see page 36 and 36).
3. Point 3 (user studies) is the focus of the next few months.
4. We'll address point 4 (linking between document types) if the need arises.
5. As to point 5 (other applications of this toolkit), we exploring what value added these tools add to Mucino's "context discovery" problem.

1. The current system was designed as a throw-away prototype to serve as an experimental workbench for numerous implementation ideas. For example, HAMLET is currently implemented in JAVA since this was a language familiar to the student programmers working on this project. The limitations of JAVA, in terms of runtime, are becoming apparent. We wish to avoid a port to "C" and are looking into better internal JAVA-based data structures.
2. As to other matters, the scalability of HAMLET's queries has yet to be demonstrated. Theoretically, the algorithms within HAMLET's current design run in linear time, or less. However, this scalability must be empirically verified.
3. Also, the user profile system that takes input from the users, then dynamically tunes the results of each query, is still being designed. Until that feature is designed, implemented, and tuned, then users of HAMLET will suffer from too many false positives. This user profile feature must be built.
4. Further, HAMLET's ability to link between documents in heterogeneous collections has yet to be tested. This test must be conducted.
5. More generally, another useful question is "what else is HAMLET good for?". HAMLET is a combination of feature extractors from technical documents, numerous data & text mining techniques, some AI tools, and some information retrieval methods (including methods for learning from user feedback). While the current version answers the questions "what else" and "what not", it is an interesting question to ask if the toolkit could be bent to another purpose.

Figure 26: Our current todo list.

References

[1] R. A. Baeza-Yates and B. Ribeiro-Neto, editors. *Modern Information Retrieval*. Addison-Wesley, 1999.

[2] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML'06*, 2006. Available from `http://hunch.net/~jl/projects/cover_tree/cover_tree.html`.

[3] G. Boetticher, T. Menzies, and T. Ostrand. The PROMISE Repository of Empirical Software Engineering Data, 2007. `http://promisedata.org/repository`.

[4] G. Booch. Software archeology, 2004. Available from `http://www.booch.com/architecture/blog/artifacts/Software\%20Archeology%.ppt`.

[5] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 475–480, New York, NY, USA, 2002. ACM.

[6] A. Dekhtyar, J. H. Hayes, and J. Larsen. Make the most of your time: How should the analyst work with automated traceability tools? In *ICSEW '07: Proceedings of the 29th International Conference on Software Engineering Workshops*, page 71, Washington, DC, USA, 2007. IEEE Computer Society.

[7] W. Dillon and M. Goldstein. *Multivariate Analysis: Methods and Applications*. Wiley-Interscience, 1984.

[8] P. Domingos and M. J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[9] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *International Conference on Machine Learning*, pages 194–202, 1995. Available from `http://www.cs.pdx.edu/~timm/dm/dougherty95supervised.pdf`.

[10] C. Faloutsos and K.-I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In M. J. Carey and D. A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 163–174, San Jose, California, 22–25 1995.

[11] B. Faltings and P. Pu. Preference-based search using example-critiquing with suggestions. *Journal of Artificial Intelligence Research*, 27:2006, 2006.

[12] C. Gupta and R. Grossman. Genic: A single pass generalized incremental algorithm for clustering. In *In SIAM Int. Conf. on Data Mining*, pages 22–24. SIAM, 2004.

[13] M. Halstead. *Elements of Software Science*. Elsevier, 1977.

[14] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Trans. Software Eng.*, 32(1):4–19, 2006.

[15] E. Hill, L. Pollock, and K. Vijay-Shanker. Exploring the neighborhood with DORA to expedite software maintenance. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 14–23, New York, NY, USA, 2007. ACM.

[16] R. Jardim-Goncalves, N. Figay, and A. Steiger-Garcao. Enabling interoperability of step application protocols at meta-data and knowledge level.

[17] T. Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In D. H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 143–151, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US. Available from `citeseer.ist.psu.edu/joachims97probabilistic.html`.

[18] T. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, Dec. 1976.

[19] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178, New York, NY, USA, 2000. ACM.

[20] J. C. Munson and T. M. Khoshgoftaar. The use of software complexity metrics in software reliability modeling. In *Proceedings of the International Symposium on Software Reliability Engineering, Austin, TX*, May 1991.

[21] M. Porter. An algorithm for suffix stripping. In K. S. Jones and P. Willet, editors, *Readings in Information Retrieval, San Francisco: Morgan Kaufmann*. 1997.

[22] J. D. M. Rennie, J. Teevan, and D. R. Karger. Tackling the poor assumptions of naive bayes text classifiers. In *In Proceedings of the Twentieth International Conference on Machine Learning*, pages 616–623, 2003.

[23] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw Hill, 1983.

[24] D. Čubranić and G. C. Murphy. Hipikat: recommending pertinent software development artifacts. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 408–418, Washington, DC, USA, 2003. IEEE Computer Society.

[25] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

[26] I. H. Witten and E. Frank. *Data mining. 2nd edition*. Morgan Kaufmann, Los Altos, US, 2005.

[27] C. Zhai. A cross-collection mixture model for comparative text mining. In *In Proceedings of KDD 04*, pages 743–748, 2004.