The HAMLET Project
Year 2, First Quarter

Tim Menzies (`tim@menzies.us`)
Greg Gay
Andrew Matheny
Adam Nelson
LCSEE, WVU

About this document: This report describes progress on the HAMLET "anything browser". It updates a prior version of this document, released Sept 2008. For a full list of all updates to the prior version, see the *status report* on the last page of this document.

Contents

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.

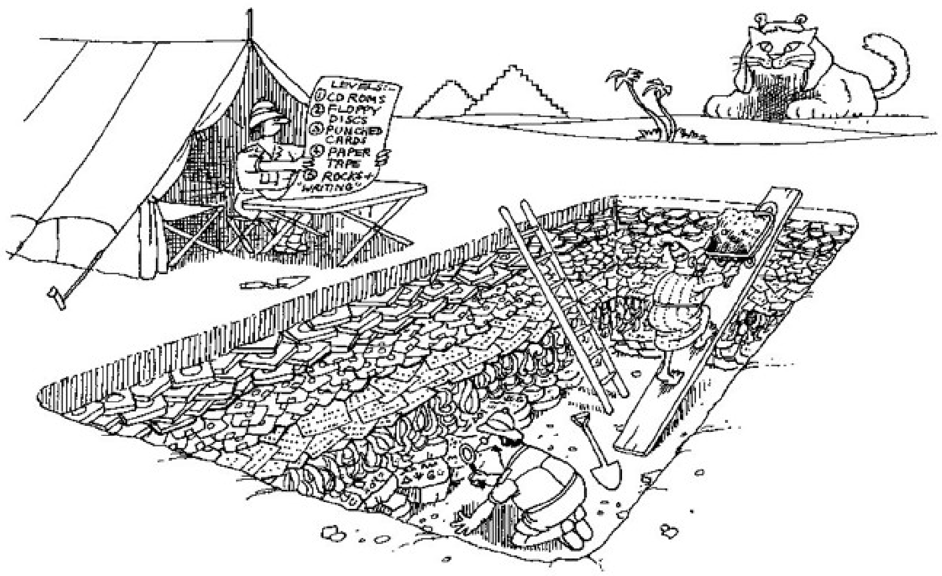


Figure 1: Digging up old designs. From [?].

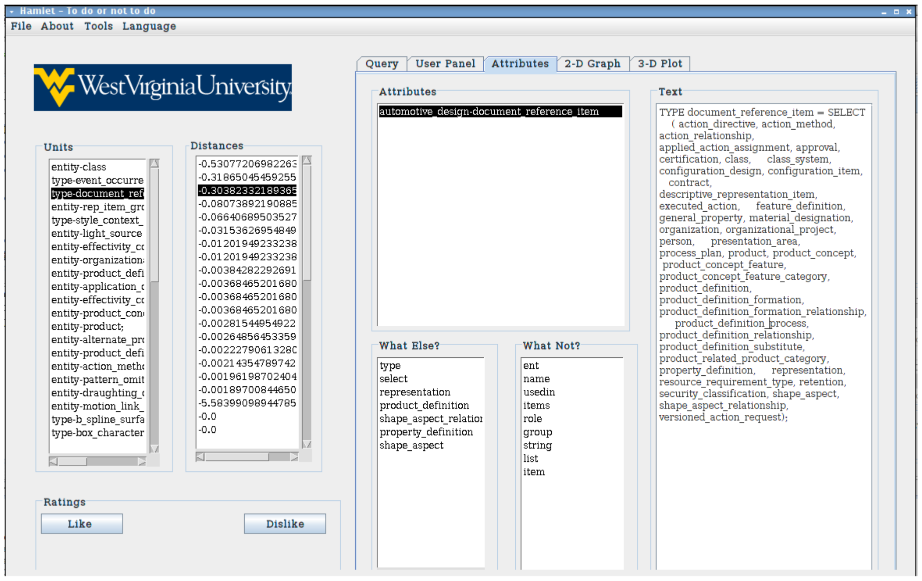
2 What is HAMLET? (overview)

2.1 What Else, What Not

HAMLET was named after the famous Shakespeare quote "to be or not to be?" For the designer of a 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 2 offers an example of these two lists.



Units	Distances
entity-class	-0.53077206982263
type-event_occurre	-0.31865045459255
type-document_ref	-0.30382332189365
entity-rep_item_grc	-0.08073892190885
type-style_context_	-0.06640689503527
entity-light_source	-0.03153626954849
entity-effectivity_cc	-0.01201949233238
entity-organization	-0.01201949233238
entity-product_defi	-0.00384282292691
entity-application_c	-0.00368465201680
entity-effectivity_cc	-0.00368465201680
entity-product_con	-0.00368465201680
entity-product;	-0.00281544954922
entity-alternate_pr	-0.00264856453359
entity-product_defi	-0.00222790613280
entity-action_meth	-0.00214354789742
entity-pattern_omit	-0.00196198702404
entity-draughting_c	-0.00189700844650
entity-motion_link_	-5.58399098944785
type-b_spline_surfa	-0.0
type-box_character	-0.0

Attributes

automotive_design-document_reference_item

What Else?

type
select
representation
product_definition
shape_aspect_relatio
property_definition
shape_aspect

What Not?

ent
name
usedin
items
role
group
string
list
item

Text

```
TYPE document_reference_item = SELECT
   ( action_directive, action_method,
action_relationship,
applied_action_assignment, approval,
certification, class,    class_system,
configuration_design, configuration_item,
  contract,
descriptive_representation_item,
executed_action,    feature_definition,
general_property, material_designation,
organization, organizational_project,
person,    presentation_area,
process_plan, product, product_concept,
 product_concept_feature,
product_concept_feature_category,
product_definition,
product_definition_formation,
product_definition_formation_relationship,
    product_definition_process,
product_definition_relationship,
product_definition_substitute,
product_related_product_category,
property_definition,    representation,
resource_requirement_type, retention,
security_classification, shape_aspect,
shape_aspect_relationship,
versioned_action_request);
```

Figure 2: A sample HAMLET screen. To the left are the nearest concepts to your new design, along with their corresponding distances. 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).

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.

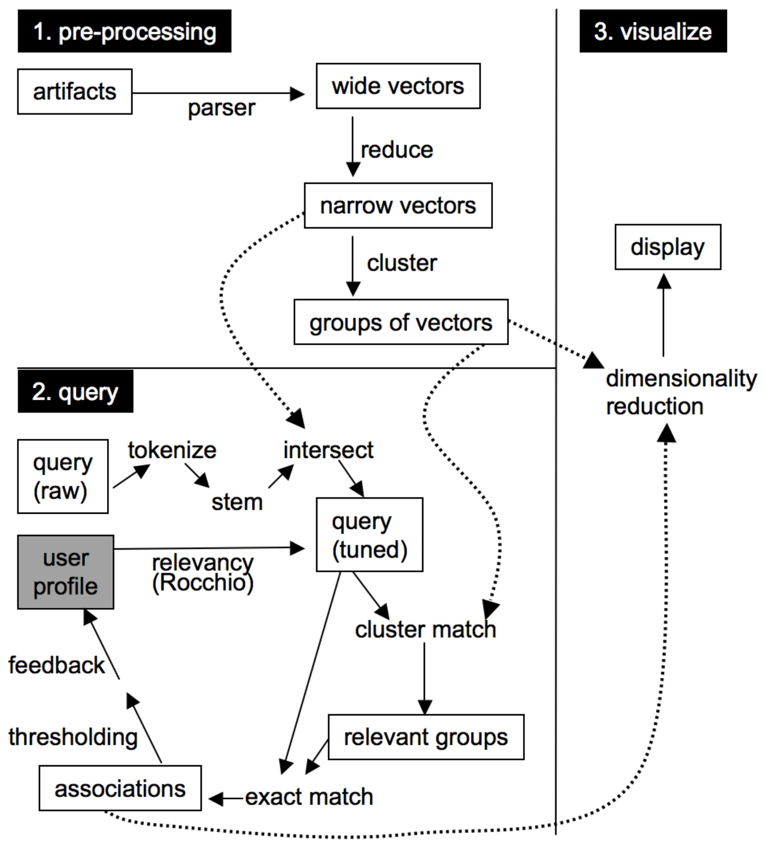


Figure 3: HAMLET's search for associations.

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 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.

2.2 In Operation

HAMLET operates in three phases (see Figure 3). All these stages have the same goal: from a large set of possible associations, extract the small subset that (a) have occurred frequently in prior technical documents and which (b) the user will approve. Stages one and two are required but stage three (visualization) is optional. Before running any query:

- Archival artifacts must be parsed into a network of nodes and edges. Each node is represented as a set of *terms* referenced in that node. This set of terms can include every term in the archive so we call these the *wide vectors*.
- The wide vectors are unwieldy to process. Hence, we run some linear time *reduction* methods to isolate the most informative columns. This produces the *narrow vectors*. Note that, when we run a query, we only use the parts of the query that appear in these narrow vectors.
- HAMLET also clusters the narrow vectors into *groups*. These groups define cliches of repeated structures.

When running a query:

- We remove irrelevant detail from the query. Linear time methods remove white space and spurious word endings. Then the query is pruned back to include just the terms seen in the narrow vectors.
- The resulting *tuned vector* is then matched to the the groups of vectors found above. For computational reasons, we run this match in two-stages. First, we find the N nearest clusters (this generates the *relevant groups*). Second, searching just within those relevant groups, we perform exact matches to find related terms.
- The *candidate matches* found from the exact match are then ranked (by the size of the overlap of the query and the terms) then pruned using thresholding (find the neighboring items in the sort with the biggest difference between them; prune the items below that largest cliff).
- These matches are then queries for “what else”. Also, the terms in the tuned query that are furtherest away from the relevant groups are the “what not”.
- The user assesses the matched queries and declares that some are “relevant” and some aren’t. This builds up a session log for this user working these kinds of queries. Once this log grows beyond a certain size, it is used to refine the tuned query such that the tuning favors nodes that do not contain what the user has labeled “irrelevancies” and does contain what the user has called “relevances”.

Optionally, we can visualize the results:

- The N-dimensions of the vectors are mapped down to 3 dimensions, then visualized on the screen.

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 [**?**].

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

- Zhai et al. [?] 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 [?] 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 [?] 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 is-a 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 [**?**] including AP-203 (for geometry) and AP-213 (for numerical control). The list of currently defined application protocols is very extensive (see Figure 4). 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 4 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.

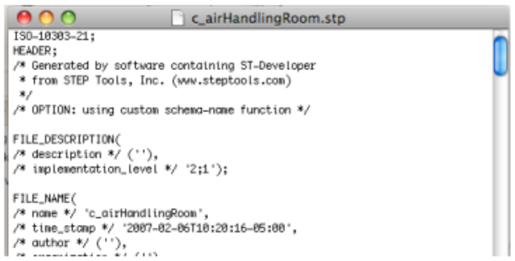
HAMLET provides an easy-to-use interface to visualize that data and make deductions based on the raw content (see Figure 5). The document panel lists only the documents that are relevant to your query (which can be a STEP design that you like). You can click on one of those documents ans view the raw 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 4: 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.

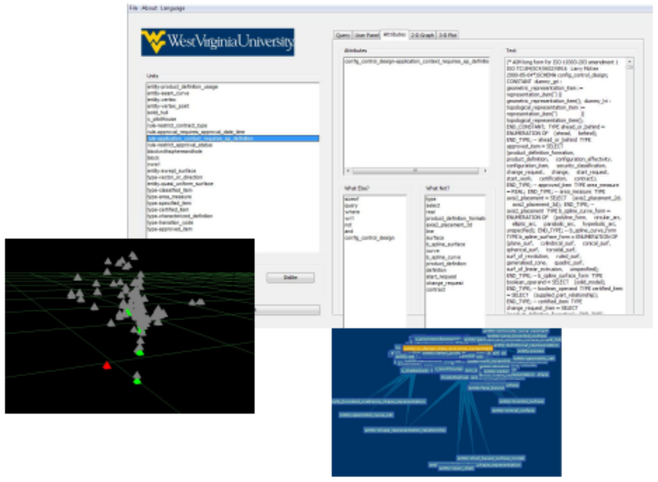


```
ISO-10303-21;
HEADER;
/* Generated by software containing ST-Developer
 * from STEP Tools, Inc. (www.steptools.com)
 */
/* OPTION: using custom schema-name function */

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 5: 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 Logging In

HAMLET collects information on each session. This session information is used to tune the query system; e.g. not to show the user results some result that they have previously indicated that they are not interested in. Figure 6 shows the log-in screen.

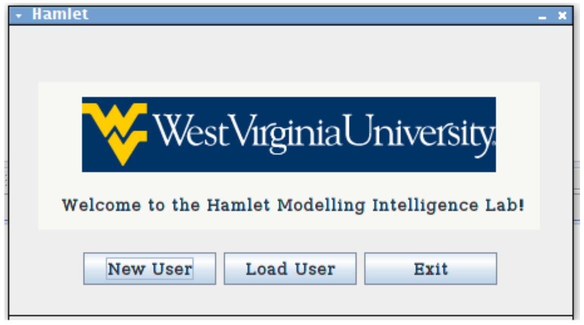


Figure 6: Logging in. The options of creating a new user or loading a user (saved preferences for a specific user) are available.

During a typical HAMLET session, the user can indicate whether they like or dislike a particular document. HAMLET utilizes information retrevial 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.

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 7: A simple user profile.

4.2 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.

HAMLET can switch between languages using the TOOLS menu of home screen of Figure 8.Upon the creation of a new user, or loading a previously created one, the default HAMLET view provides a variety of options including loading a dataset, creating a new member, altering the query language, etc.

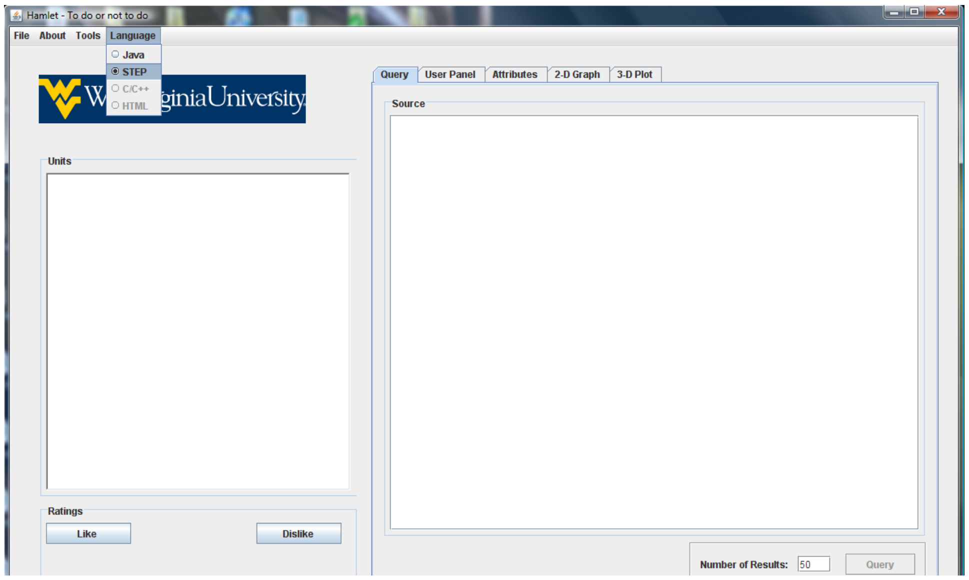
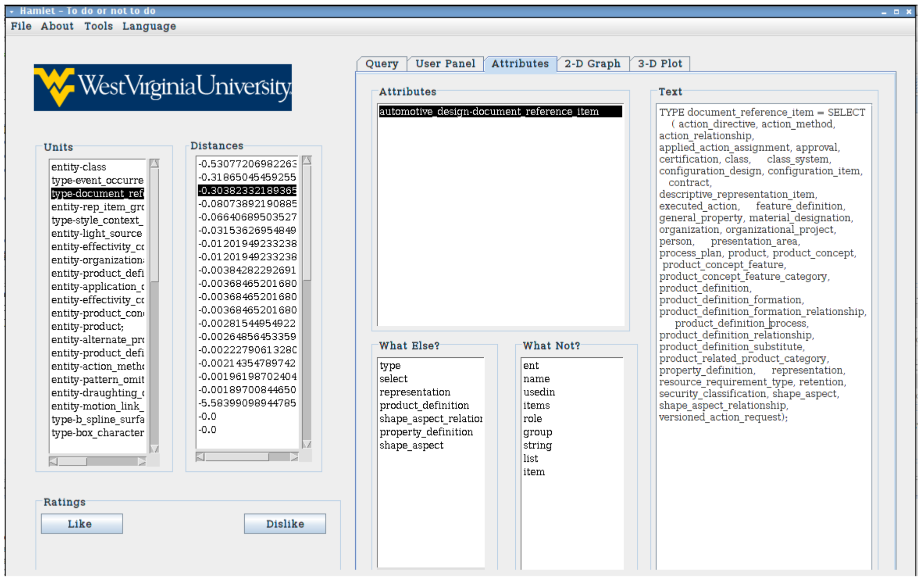


Figure 8: Altering the query language.

4.3 HAMLET output


Hamlet - To do or not to do
File About Tools Language
Query User Panel Attributes 2-D Graph 3-D Plot
Ratings
Like
Dislike

Units

```
entity-class
type-event_occurre
type-document_ref
entity-rep_item_gr
type-style_context_
entity-light_source
entity-effectivity_c
entity-organization
entity-product_defi
entity-application_
entity-effectivity_c
entity-product_con
entity-product;
entity-alternate_pr
entity-product_defi
entity-action_meth
entity-pattern_omit
entity-draughting_
entity-motion_link_
type-b_spline_surfa
type-box_character
```

Distances

```
-0.53077206982263
-0.31865045459255
-0.30382332189365
-0.08073892190885
-0.06640689503527
-0.03153626954849
-0.01201949233238
-0.01201949233238
-0.0038428229269l
-0.00368465201680
-0.00368465201680
-0.00368465201680
-0.00281544954922
-0.00264856453359
-0.00222790613280
-0.00214354789742
-0.00196198702404
-0.00189700844650
-5.58399098944785
-0.0
-0.0
```

Attributes

```
automotive_design-document_reference_item
```

What Else?

```
type
select
representation
product_definition
shape_aspect_relatio
property_definition
shape_aspect
```

What Not?

```
ent
name
usedin
items
role
group
string
list
item
```

Text

```
TYPE document_reference_item = SELECT
   ( action_directive, action_method,
action_relationship,
applied_action_assignment, approval,
certification, class,     class_system,
configuration_design, configuration_item,
  contract,
descriptive_representation_item,
executed_action,     feature_definition,
general_property, material_designation,
organization, organizational_project,
person,     presentation_area,
process_plan, product, product_concept,
 product_concept_feature,
product_concept_feature_category,
product_definition,
product_definition_formation,
product_definition_formation_relationship,
    product_definition_process,
product_definition_relationship,
product_definition_substitute,
product_related_product_category,
property_definition,     representation,
resource_requirement_type, retention,
security_classification, shape_aspect,
shape_aspect_relationship,
versioned_action_request);
```

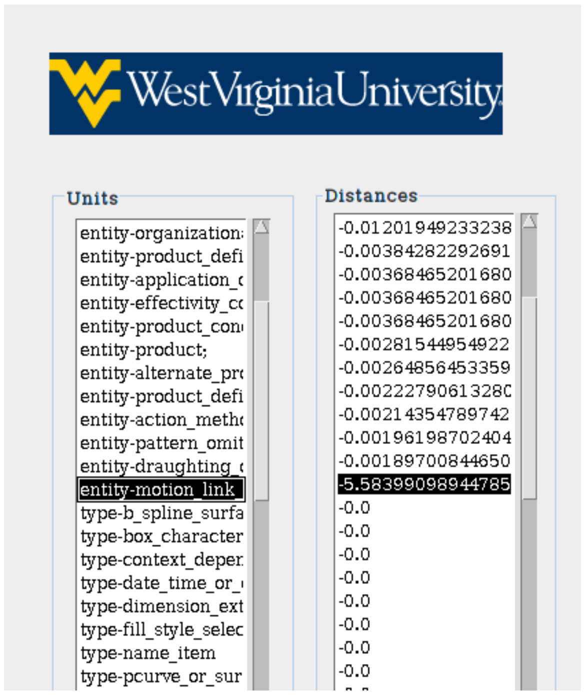
Figure 9: A sample HAMLET screen

Figure 9 shows HAMLET, after running a query. To the left are the nearest concepts to your new design, along with their corresponding distances. 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).

An additional function of HAMLET is to provide the user with the ability to view document vector distances. In order to retain order, when a user clicks on a document in the unit list, that particular unit's distance is immediately highlighted (see Figure 10). The inverse is also possible, by first selecting a distance, which shows the user the corresponding document.

By selecting which vectors to visualize, a finer grasp of relevant/irrelevant concepts can be obtained. In the above image, by selecting ”Show Close”, a 3-D visualization of the nearest related documents is shown, while ”Show Other” and ”Show Query” are used to visualize less relevant points as well as new designs, respectively (see Figure 11). These points are made clear by varying colors.

Figures 11,12,13,14 illustrate HAMLET's user feedback loop, in action. Figure 12 represents the initial state when HAMLET is started and the dataset is loaded. A query is entered by the user. In this case, the query is a bug report for HAMLET itself (i.e. numoerus JAVA classes). At this point, no user preferences have been recorded.



Units	Distances
entity-organization	-0.01201949233238
entity-product_defi	-0.00384282292691
entity-application_	-0.00368465201680
entity-effectivity_cc	-0.00368465201680
entity-product_con	-0.00368465201680
entity-product;	-0.00281544954922
entity-alternate_pr	-0.00264856453359
entity-product_defi	-0.00222790613280
entity-action_meth	-0.00214354789742
entity-pattern_omit	-0.00196198702404
entity-draughting_	-0.00189700844650
entity-motion_link	-5.58399098944785
type-b_spline_surfa	-0.0
type-box_character	-0.0
type-context_deper	-0.0
type-date_time_or_	-0.0
type-dimension_ext	-0.0
type-fill_style_selec	-0.0
type-name_item	-0.0
type-pcurve_or_sur	-0.0

Figure 10: Viewing document distances.

Therefore, the results in the left panel are based solely on the text content of the query.

At the point depicted in Figure 13, several items have been rated as ”liked” or ”disliked” by the user. The query has been re-run with these preferences in mind, and a new set of results has been generated. As you can see in the left panel, results that have been recorded during previous runs are marked with a + (helpful) or a - (unhelpful). The total number of returned results is displayed at the bottom, along with counts of old liked and old disliked results that are still considered to be nearby.

As shown in Figure 14, a seperate tab displays a complete list of rated documents. The number next to the name is the vertex’s unique identifier in the dataset’s graph. A set of buttons along the bottom let you ”rerate” a document or reset it to neutral as desired.

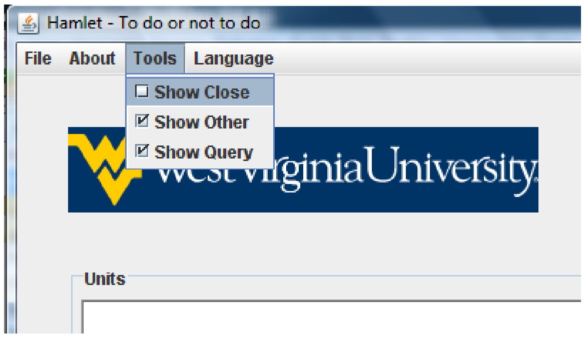
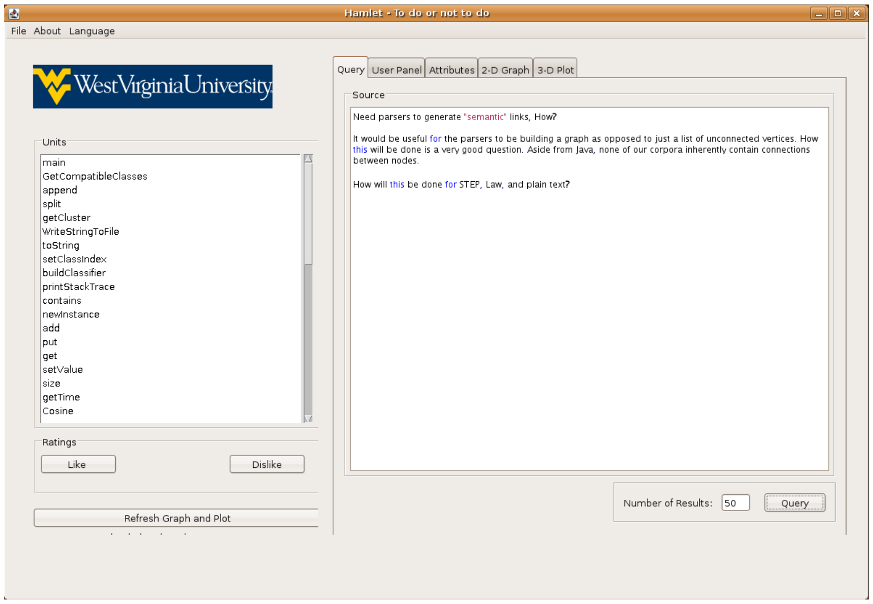


Figure 11: Select Vectors



Source

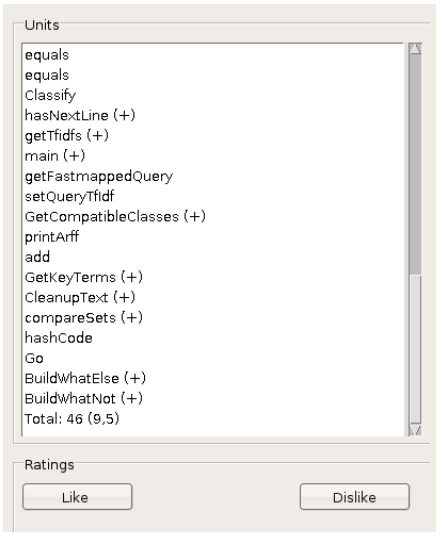
Need parsers to generate "semantic" links, How?

It would be useful for the parsers to be building a graph as opposed to just a list of unconnected vertices. How this will be done is a very good question. Aside from Java, none of our corpora inherently contain connections between nodes.

How will this be done for STEP, Law, and plain text?

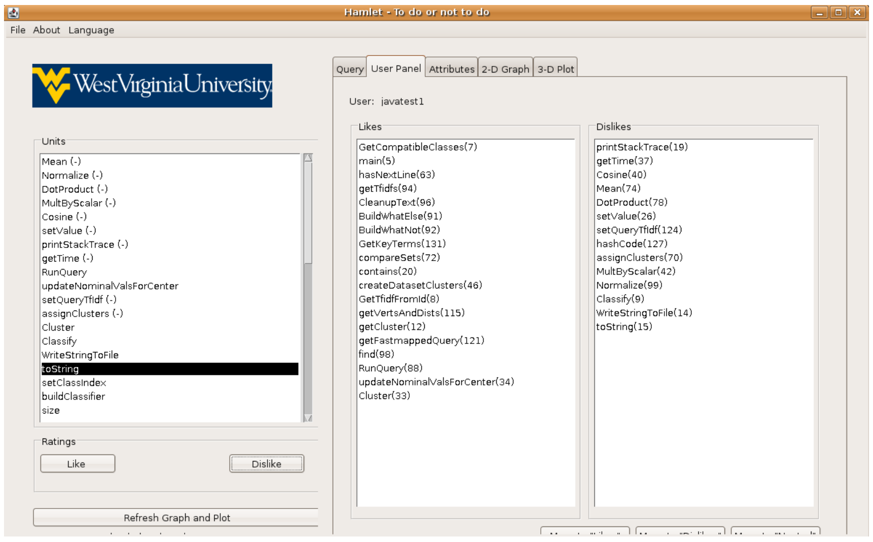
```
main
GetCompatibleClasses
append
split
getCluster
WriteStringToFile
toString
setClassIndex
buildClassifier
printStackTrace
contains
newInstance
add
put
get
setValue
size
getTime
Cosine
```

Figure 12: Initial Query Run



```
equals
equals
Classify
hasNextLine (+)
getTfidfs (+)
main (+)
getFastmappedQuery
setQueryTfIdf
GetCompatibleClasses (+)
printArff
add
GetKeyTerms (+)
CleanupText (+)
compareSets (+)
hashCode
Go
BuildWhatElse (+)
BuildWhatNot (+)
Total: 46 (9,5)
```

Figure 13: Results after items have been rated.

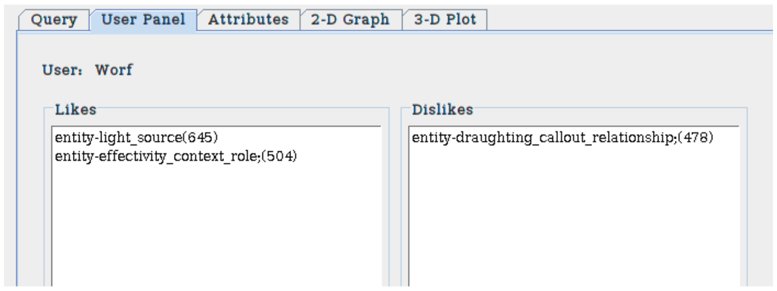


User: javatest1

```
Mean (-)
Normalize (-)
DotProduct (-)
MultByScalar (-)
Cosine (-)
setValue (-)
printStackTrace (-)
getTime (-)
RunQuery
updateNominalValsForCenter
setQueryTfIdf (-)
assignClusters (-)
Cluster
Classify
WriteStringToFile
toString
setClassIndex
buildClassifier
size
```

Likes	Dislikes
GetCompatibleClasses(7)	printStackTrace(19)
main(5)	getTime(37)
hasNextLine(63)	Cosine(40)
getTfidfs(94)	Mean(74)
CleanupText(96)	DotProduct(78)
BuildWhatElse(91)	setValue(26)
BuildWhatNot(92)	setQueryTfIdf(124)
GetKeyTerms(131)	hashCode(127)
compareSets(72)	assignClusters(70)
contains(20)	MultByScalar(42)
createDatasetClusters(46)	Normalize(99)
GetTfidfFromId(8)	Classify(9)
getVertsAndDists(115)	WriteStringToFile(14)
getCluster(12)	toString(15)
getFastmappedQuery(121)	
find(98)	
RunQuery(88)	
updateNominalValsForCenter(34)	
Cluster(33)	

Figure 14: User Preferences Tab



User: Worf

Likes

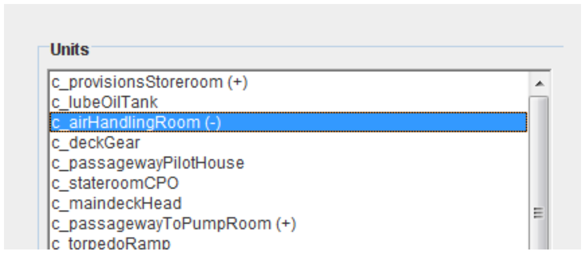
entity-light_source(645)
entity-effectivity_context_role;(504)

Dislikes

entity-draughting_callout_relationship;(478)

Figure 15: Ratings.

Figure 15 shows the rating system. Once a user rates a document, the immediate effects can be seen in the unit list. A ’Like’ is denoted as a ’(+)’, and ’Dislike’ as a ’(-)’. Also, in the User Panel, the current likes and dislikes are organized into separate lists.



Units

c_provisionsStoreroom (+)
c_lubeOilTank
c_airHandlingRoom (-)
c_deckGear
c_passagewayPilotHouse
c_stateroomCPO
c_maindeckHead
c_passagewayToPumpRoom (+)
c_torpedoRamp

Figure 16: After rating

.

As a user rates a document, all of the ratings appear in the User Panel. As shown in Figure 16, the option is given to move a document from any rating to another by highlighting the unit and clicking the appropriate button.

4.4 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.

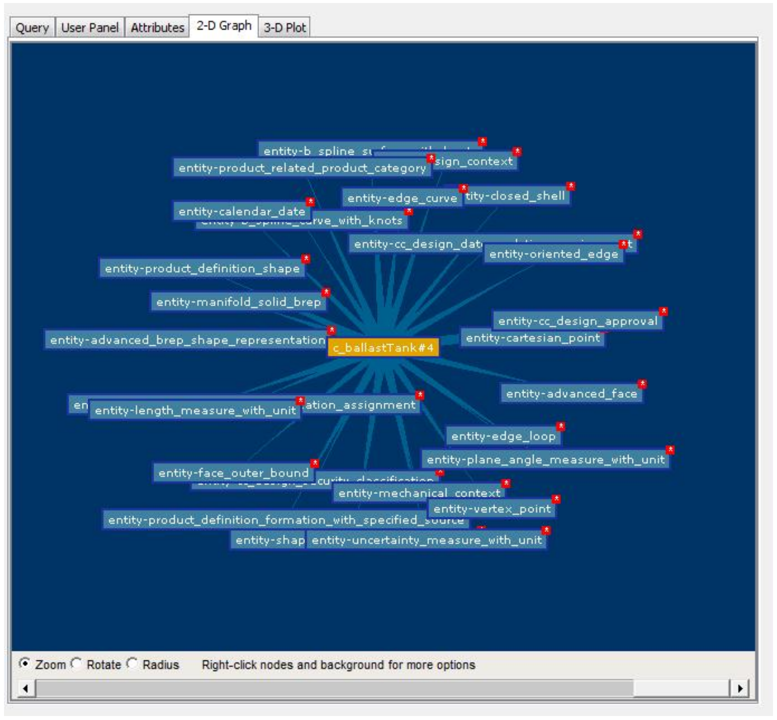

Query
User Panel
Attributes
2-D Graph
3-D Plot
entity-product_related_product_category
entity-calendar_date
entity-edge_curve
entity-closed_shell
entity-cc_design_date
entity-oriented_edge
entity-product_definition_shape
entity-manifold_solid_brep
entity-cc_design_approval
entity-cartesian_point
entity-advanced_brep_shape_representation
c_ballastTank#4
entity-advanced_face
entity-length_measure_with_unit
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-uncertainty_measure_with_unit
Zoom
Rotate
Radius
Right-click nodes and background for more options

Figure 17: Displaying connections.

4.5 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.

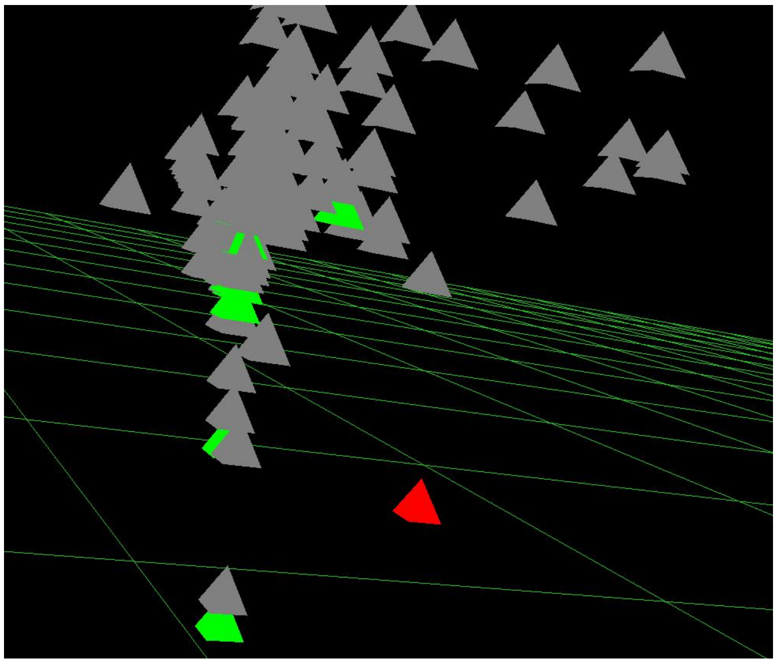


Figure 18: 3-D visualization. A new design (in red) floats near its nearest related concepts (in green). The gray points show parts of specifications that are less relevant to the new design. Note that this is a 2-D visualization of a 100-D space.

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:

- We are experiment with replacing all slower-than-linear algorithms with linear-time algorithms (see the GENIC experiments, below).
- 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:

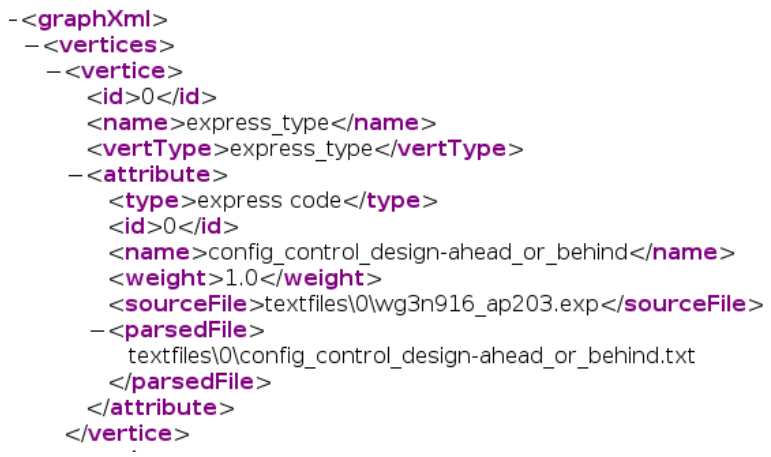
- Their relevance to this project- so STEP is highest;
- As well the availability of large corpora- so we use JAVA class libraries and a large XML dump pf 400 years of Florida real estate law.

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.

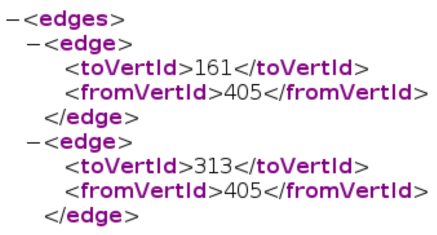
The most important function of the API is the generation of the GraphXML file. This file is the intermediary between the data set and HAMLET. It contains a list of each document (vertice) and the relationships between them. Other pertinent information, such as file pointers and document statistics, is 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 19: 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 20: 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 [13]. 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, 12]. 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 21 shows a sample of the stop list used in HAMLET. Figure 21 shows code for a stop-list function.

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

```
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 21: 24 of the 262 stop words used in this study.

- CONNECTING
- CONNECTION
- CONNECTIONS

Porter's stemming algorithm [11] 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 22).

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 22: 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

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

"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:

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

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.

$$InfoGain(A_i) = H(C) - H(C|A_i) \quad (4)$$

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:

$$tr_i = \sum_{d \in D} TfIdf_{i,d} \quad (5)$$

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 23 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.

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:

[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 [6].

```
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 23: 30 of the 100 key terms found in a STEP dataset using TF*IDF ranking

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) [4] has been widely applied to resolve problems with structural code measurements; e.g. [10]. 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 24 (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

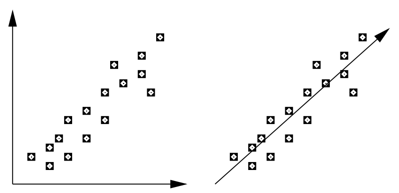


Figure 24: The two features in the left plot can be transferred to the right plot via one latent feature.

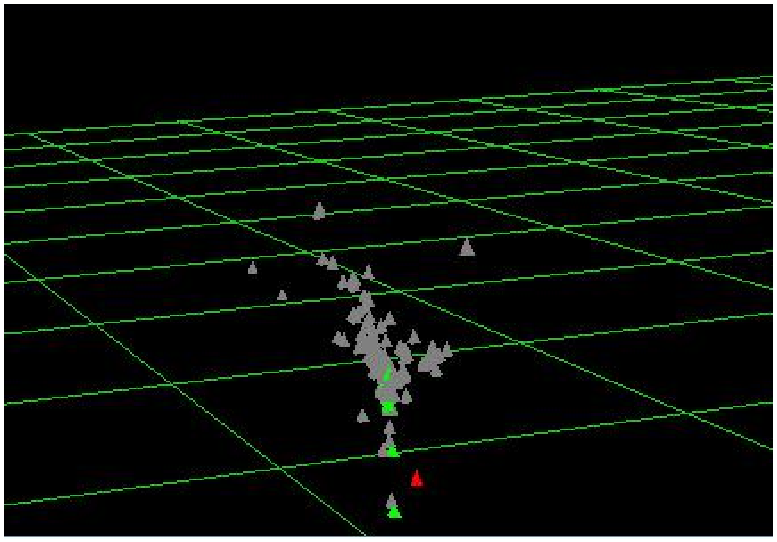


Figure 25: A 100-D space presented in 3-D. The point in red shows the current design and the four points in green show the nearest 4 technical documents (and the gray points show other documents that have been "trimmed" away using the techniques discussed below).

or Halstead static code metrics [9, 7] 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 25 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 [**?**]. Our own experiments with the two methods showed that both yield similar structures.

5.4 Clustering

The goal of HAMLET, as defined thus far, is to index large amounts of technical information spread across a variety of document types in a manner that allows us to ask the question, ”What designs are their similar to this design?” In order be able to answer this question we must first find an appropriate way of locating the ”structure” in these collections; including (a) the structure that exists within each document as part of the collection, and (b) the comparative structure of all the documents and how they relate to one other. Considering that each document type has a format that can vary from highly syntactic source code, to badly formed HTML, to unstructured text, there is not much common ground between the various document types. One common theme in all of these documents is the exact thing you are looking at right now, natural language. By grouping all the natural language associated with a single design into one document we can solve some of the problems. This answers (a), the question of how to define the structure of each document as its own entity within the larger collection, but not (b) the question of how to define the comparative structure of the documents and how they relate to each other. Enter document clustering.

By clustering text documents we are able to find the inter-document structure in the collection. This is done by locating groupings of documents, which can later be used to give us intelligent advice on what designs are similar. Below we illustrated a few common clustering methods and how they might proof useful in the context of HAMLET. In later sections of this document, we provide emperical analysis of two of the algorithms presented, GENIC and K-Means.

5.4.1 CLustering with GENIC

GENIC is a generalized incremental clustering algorithm developed by Gupta and Grossman [**?**] that provides potentials for large improvements in scalability over K-Means. 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 when dealing with large corpora. By using stochastic methods, GENIC can be given an initial k equal to the number of items (each item is its own clusters) and 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 is of specific interest to HAMLET for two primary reasons, low expected run-times on large corpora and a potential ability at estimating the number of natural clusters in the collection.

- 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: Because of GENIC's stochastic based method of removing unwanted or non-useful clusters, it has potential for use in correctly estimating a good value for k. By eliminating "bad stuff", GENIC can ideally identify the correct number of types of "good stuff".

5.4.2 Clustering with 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 26. For an example of the algorithm in operation, see Figure 27.

- i=0
- Partitioning the input points into k initial sets, either at random or using some heuristic data.
- Repeat unitl ($i \leq maxIterations$ or no point changes set membership)
  - 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

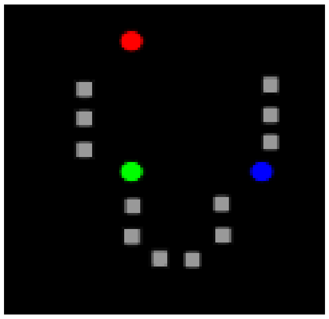
Figure 26: K-Means algorithm. See Figure 27 for an example of this algorithm running in practice.

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

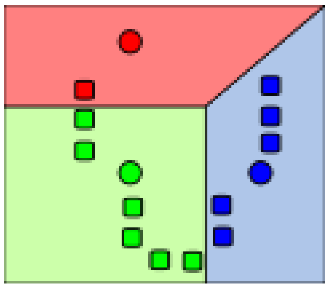
Another problem with k-means is determining what value of k should be used. Note the usability issues with requiring a user to pre-specify k: isn't this the kind of tedious detail that the computer should be telling us?

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. In the sequel, we will empirically evaluate the value of GENIC's random seaerch vs K-means slower search.

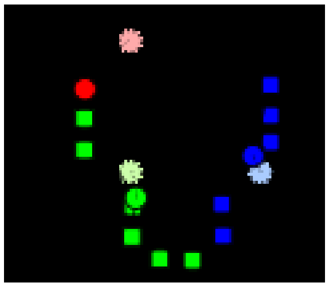
Step1: Here, we show some initial data points and the centroids generated based on random assignment



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.

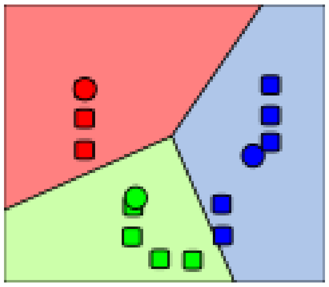


Figure 27: Example of K-means

5.4.3 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* [**?**, **?**]. 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 28);
- 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.

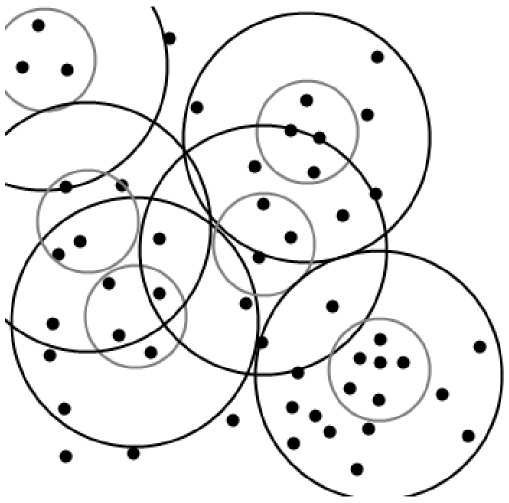


Figure 28: The darker circle represents all points in a given canopy, points in the smaller circles cannot be used as a new canopy center.

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 [14]:

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

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 [5]. On average, Naive Bayes classifiers perform as well, if not better than, more complex classification algorithms.

5.5.2 TWCNB

Rennie et al. [**?**] 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:

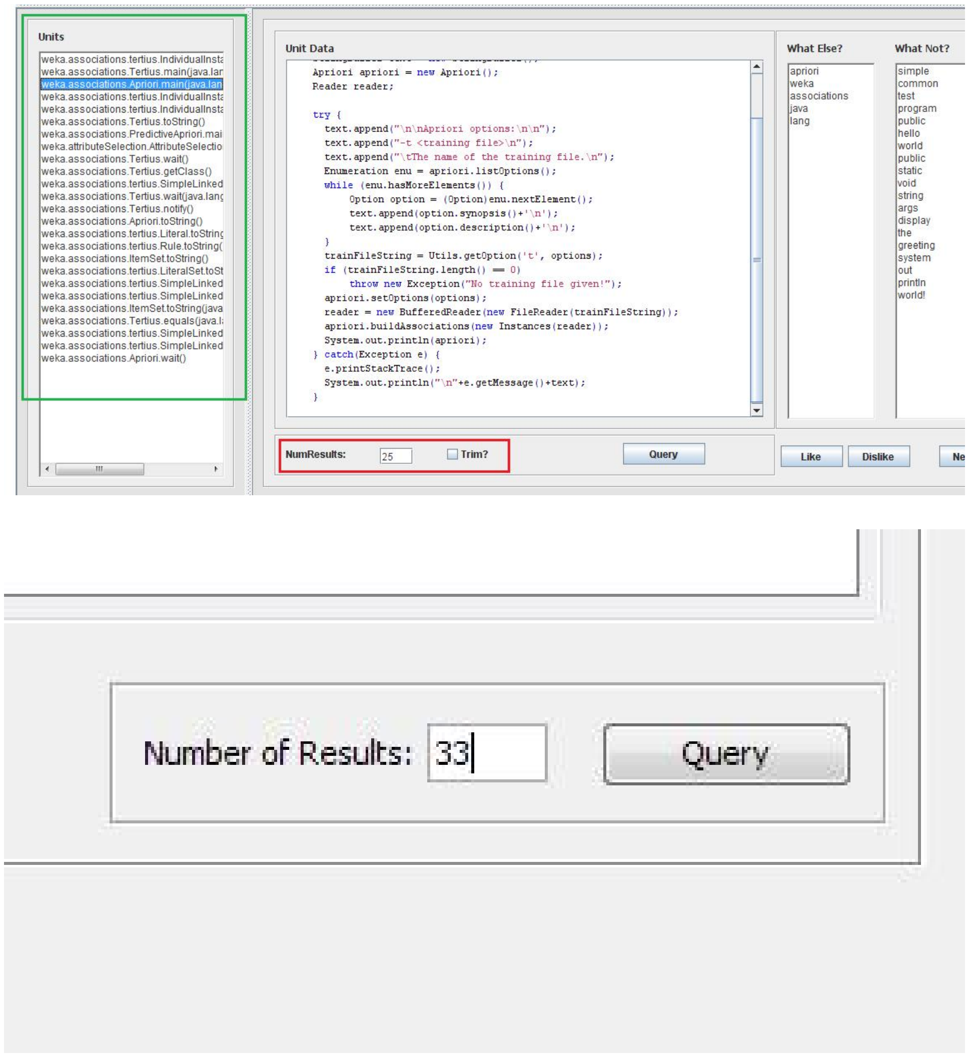
- The user specifies some max distance measure N.
- The N nearest documents to the user's partially specified design are accessed. These documents are sorted $1, 2, 3, 4...N$ according to their distance to the current design.
- The delta between document i and $i + 1$ in the sort order is computed and 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 29 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



weka.associations.tertius.IndividualInsta
weka.associations.Tertius.main(java.lar
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.l
weka.associations.tertius.SimpleLinked
weka.associations.tertius.SimpleLinked
weka.associations.Apriori.wait()

```
    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!

NumResults: 25 ☐ Trim?

Number of Results: 33

Figure 29: 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.

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).
- 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.

5.8 User Profiling with the Rochio Algorithm

5.8.1 Motivation

Without any sort of user feedback mechanism, the use of HAMLET is a one-time static process. The user will enter their input design and look at the space of similar objects. Some of those similar artifacts will be useful in the design process. However, it is equally likely that some of these ”similar” objects will be unhelpful. This is why some form of a user feedback mechanism could be helpful.

A user feedback mechanism could make this static process into a repeatable interactive one. The preferences expressed during one stage of the design process could bias future iterations of the same design. During a single session, a query could be run over and over, with the user rating additional items each time.

As a user searches the space of similar artifacts, they could mark these items as ”liked” or ”disliked.” This set of preferences could be used on subsequent iterations of the same query to bias the results. In HAMLET, the Rochio user feedback formula is used to move the point in space that the query represents to another spot. Ideally, this new location is ”closer” to more relevant results. As more items are rated, this point will continue to move within the space of document results.

5.8.2 What is the Rochio Formula?

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 [8, **?**] 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 [**?**]. They are then divided by the size of each set. These summations are then used to tune the original query vector as follows:

$$Q_n = \alpha Q_{n-1} + \frac{\beta}{|L_q|} \sum_{d \in L_q} d - \frac{\chi}{|U_q|} \sum_{d \in U_q} d \tag{13}$$

According to Dekhtyar et al [**?**], 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).

This equation is recursive in nature. When a new query is supplied by the user, the formula uses that as Q_{n-1}. The resulting query, Q_n becomes Q_{n-1} during the next iteration. Each round, the point in space moves slightly more in a direction taken from the centroid of the liked documents and the centroid of the disliked documents.

The Rochio formula has a set of weighted constants that can be used to place importance on the old query, the liked items, or the disliked items. Although certain past research [**?**] suggests a weight of 4 on liked items and 16 on disliked items, others [**?**] have found little practical use for them. By default, these weights are all set to ”1” in HAMLET. We are experimenting with different treatments in order to find the most effective one. Placing twice the importance on the old query (a setting of 2,1,1 for the constants) has showm promise, but most complicated trials have not been performed yet.

Essentially, all that the Rochio formula does is move your position in space. Ideally, it will move towards more helpful documents, but the burden for this falls on the clusterer and the clustering method used. If a cluster contain both helpful and unhelpful documents, both types will appear in your results as the query moves closer to it. Although it would be impossible to generate perfect clusters. Whether a document is helpful or not depends on the current task. However, it is likely that the helpful items will be similar to each other. If a cluster contains extemely similar documents, it might be more likely that those documents will all be helpful. Improved clustering performance could yield better Rochio results. Future work on both the user feedback mechanisms and the clustering will go hand-in-hand.

5.8.3 HAMLET and Rochio

Although the Rochio formula has not been used in prior HAMLET-like systems, it seemed to be the most appropriate user feedback mechanism for the purposes of this project. The Rochio formula is a standard technique in the information retrieval field because it is based on the concept of TF*IDF, which is a concept used heavily within HAMLET. Rochio has yielded strong results in systems that share common IR and machine learning techniques with HAMLET. It is for this reason that Rochio was chosen as the initial user feedback mechanism.

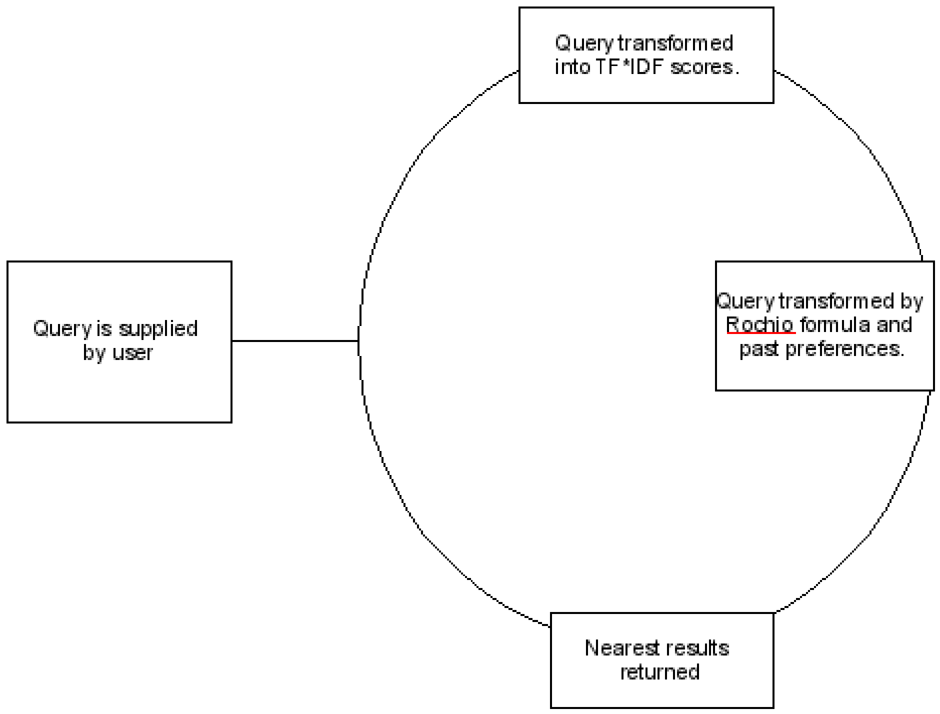

Query transformed into TF*IDF scores.
Query is supplied by user
Query transformed by Rochio formula and past preferences.
Nearest results returned

Figure 30: The user-feedback cycle

This chart illustrates the user feedback cycle during a session with HAMLET. The user supplies a query in text form in the HAMLET browser window. Once the run button is clicked, this query is cleaned up and any unnecessary terms are removed. The TF*IDF scores are calculated and normalized for the remaining terms. At this point, the Rochio formula is applied to the user's query as explained in the previous section. This modified query is saved in memory then passed off to the classifier, which returns a set of results.

6 Clustering Experiments

In this section we describe empirical analysis into the trade-offs of GENIC and K-means which respect to clustering textual documents in HAMLET's domain. The datasets used in these trials are of specific interest to HAMLET as they are generated from collections of STEP/EXPRESS source code. This experiment evaluates the value of the GENIC (§5.4.1) and K-means (§5.4.2) for clustering archival data. K-means is a standard clsutering algorithm and GENIC is a novel stochastic method that showed much promise.

After the experiments described below, we now recommend against GENIC for the following reason. Some trade-offs are acceptable. In some cases, we would accept that trade-off (e.g. a 10% reduction in clustering quality in exchange for the ability to examine data sets 10 times as big). However, as reported below, the trade-off between GENIC and K-means is unacceptable.

6.1 Experimental Design

In order to successfully measure the merits of GENIC and K-means, an evaluation criteria must first be defined. The two criteria we care most about are, ”how good is the clustering result?” and ”How long did it take us to come up with?” The second question can be answered by simply measuring the run-times of the algorithms for the same problem. The first question however needs slightly more effort in order to come up with a good comparison mechanism.

For analyzing clustering accuracy or validity, evaluation metrics can be grouped into two categories, external and internal [**?**]. External quality is based upon how the clusters output match up with some initial correct labeling, while internal quality is based entirely on the clusters output with no correct results to judge against. For the purposes of this experiment, we can only focus on internal cluster quality since the information needed for external quality does not exist for the text collections being used in this study. To evaluate cluster accuracy, we are using two measures of internal cluster validity, intra-cluster similarity and inter-cluster similarity [**?**].

The intra-cluster similarity is effectively the pairwise similarity between all elements within a cluster If n_j is the number of elements in cluster j, and N is the total number of elements in the corpus then intra-cluster similarity is defined as follows:

$$ISim_j = \frac{1}{n_j^2} \sum_{d \in C_j d' \in C_j} cos(d, d') \tag{14}$$

$$Intra\ cluster\ similarity = \sum_j \frac{n_j}{N} ISim_j \tag{15}$$

Using the same notation, the inter-cluster similarity is defined as follows:

$$ESim_i j = \frac{1}{n_i n_j} \sum_{ainC_i, binC_j} cos(a, b) \tag{16}$$

$$Inter\ cluster\ similarity = \sum_j \frac{n_j}{N} ISim_j \tag{17}$$

The different treatments involved in the experiment will be all pairs of levels the two independent factors, clusterers GENIC, K-means and a collection of values for $k \in 1, 4, 16, 32, 64, 128, 256, 512, 1024$. Because both K-means and GENIC depend on stochastic methods, we have repeated the experiment 20 times for each treatment to minimize possible glitches caused by randomness. Furthermore, the random seed was based on time to ensure new random numbers were being generated for eah repeat to void seeing results that can be attributed to the effects of a non-randomized process.

The datasets being used for the experiment are two text datasets generated from collections of STEP/EXPRESS documents. The STEP/EXPRESS files were separated based on which AP they conform to/describe, resulting in two datasets; one from AP203 and the other AP214. Further details are described in Figure 31

	AP203	AP214
Total Documents	484	1373
Total Terms	1103	3050
Mean Document Length	143	164
StdDev of Document Length	8750	2313

Figure 31: Statistics on STEP/EXPRESS datasets.

6.2 Previous Works

Gupta & Grossman compare GENIC vs. windowed K-Means in [**?**] on metrics regarding run-time and cluster quality using Sum of Squares of Errors or Inter-cluster similarity as their choice for the quality feature. Their results show GENIC and K-Means very close for all cases except when GENIC was given a very large number of centers to start with. However, all that was given in their were a few graphs and subtle commentary. No statistical analysis was done on their results of either run-time or cluster quality leaving room for more work to be done in this area. Additionally, the datasets used by Gupta & Grossman in this paper were all artificially created, which may raise questions about the external validity of their findings. With respect to the run-time results of this study, they found GENIC running a maximum of 40 times faster than K-Means, however this was only a maximum value that was given with no further statistical results.

6.3 Discussion of Results

6.3.1 Run-time

Which respect to run-times, the results were largely as expected. K-Means is not designed for high speed scenarios like GENIC is, and the results show this. What is surprising however, is that the low magnitude of victory (see K-Means/GENIC in Figure 32). Contrary to evidence presented by Gupta in 2004, on average GENIC's run-time was only 10.4 times slower than K-Means (compared to 40 times faster as presented by Gupta). This is even more surprising given the previous comparison was against a version of K-means which was optimized for speed.

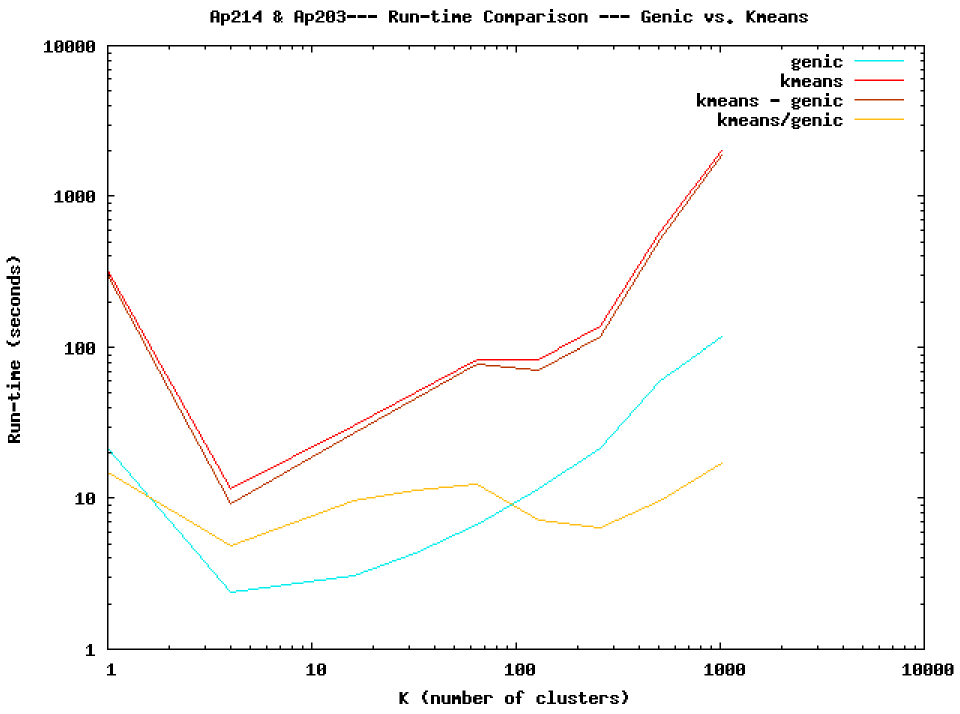


Figure 32: Average run-time graph of GENIC, K-Means, K-Means - GENIC, and K-Means/GENIC

For the analysis of run-times, Mann-Whitney U tests were used because of their non-parametric nature (run-times may not necessarily conform to a Gaussian distribution). By looking at the results from the MWU tests in Figure 33 below, you will see that GENIC consistently out performs K-Means with respect to run-times, at the 95% confidence interval. Given the linear approach of GENIC, this is as expected.

treatment	ties	win	loss	win-loss-at-95%
genic & k=4	1	15	1	14
genic & k=1	1	15	1	14
genic & k=16	0	14	3	11
genic & k=32	0	13	4	9
genic & k=64	0	12	5	7
kmeans & k=4	1	10	6	4
genic & k=128	1	10	6	4
kmeans & k=16	1	8	8	0
genic & k=256	1	8	8	0
kmeans & k=32	1	6	10	-4
kmeans & k=64	2	4	11	-7
genic & k=512	4	3	10	-7
kmeans & k=128	3	3	11	-8
kmeans & k=256	2	2	13	-11
genic & k=1024	2	2	13	-11
kmeans & k=512	0	1	16	-15
kmeans & k=1024	0	0	17	-17

Figure 33: Mann-Whitney U tests for run-time of GENIC and K-Means. As expected, GENIC wins over all K-means treatments within a few steps of *k*.

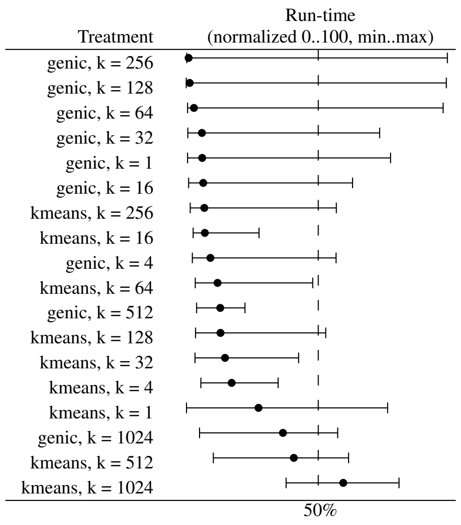


Figure 34: Run-times for GENIC and K-Means with different values for *k* (25 to 75% percentile range from 40 simulations; median values shown as a dot). As expected, GENIC outperforms K-Means in nearly all treatments.

6.3.2 Cluster Validity

When it comes to the validity of the clusters generated by GENIC and K-Means, the results are completely opposite from the run-time results. Figure 38 shows the Mann-Whitney results for intra-cluster similarity. In the MWU tests shown in Figure 38

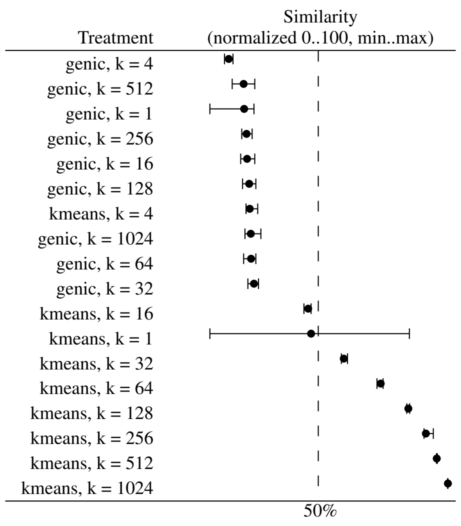


Figure 35: Cluster similarity measures using GENIC and K-Means with different values for *k* (25 to 75% percentile range from 40 simulations; median values shown as a dot). Surprisingly, GENIC does not perform well at all with respect to cluster similarity.

GENIC does not have a single treatment that scores more wins than losses at the 95% significant level.

This could be attributed to the stochastic removal of clusters done by GENIC. However, K-Means also uses randomness when computing its initial clusters. One may also consider the fact that GENIC does not retain the full number of clusters asked for a reason for the poor performance. Figure 36 lists the actual number of clusters returned by GENIC on average. So in all actuality, when comparing GENIC with k=1024 we shouldn't compare it against K-Means with k=1024, but rather a K-Means treatment closer the value actually returned by GENIC, which is approximately 316. Though, despite this caveat GENIC still fails to beat any K-Means treatment except when k=1.

Size	**Average k actually returned**
1	1
4	3.9375
16	11.7875
32	20.275
64	35.5375
128	64.4375
256	114.013
512	189.15
1024	316.775

Figure 36: Average number of clusters returned by GENIC for each K

Looking at Figure 37 further illustrates the vast differences in similarity between K-Means and GENIC. Notice the K-means asymptotic approach to perfect similarity while GENIC stays approximately the same regardless of the number of clusters without ever raising over 0.5. Figure 37 also suggests that there is a useful maximum number of clusters in this domain: no improvement is apparent over (around) 300 clusters for either of our methods.

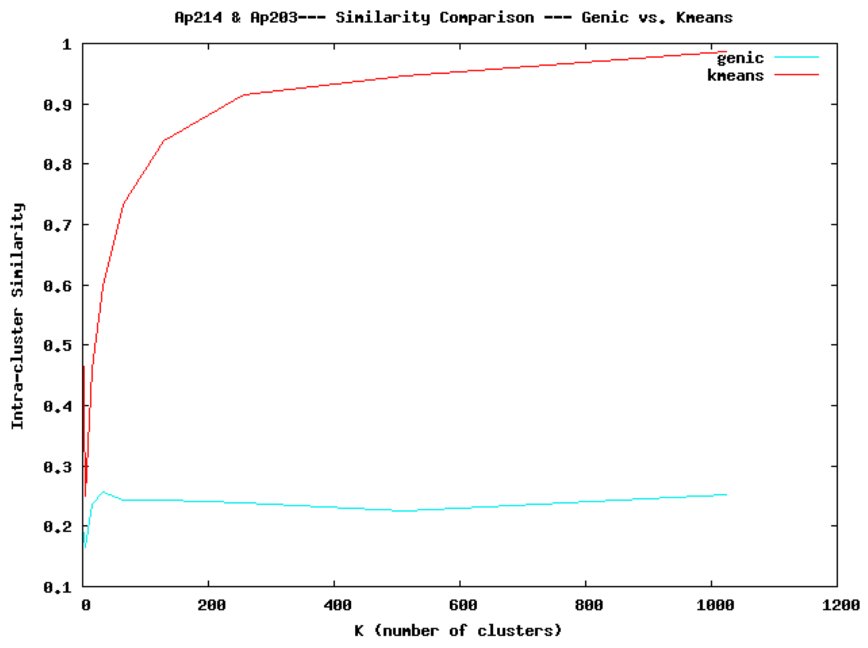


Figure 37: Intra-cluster similarity graph of GENIC and K-Means on average

It would not be as significant if only the average similarity of GENIC was so low. Given the randomness of the algorithm, its reasonable to believe that a good deal of variance could be introduced. However, looking at the quartile charts in Figure 35, you can see the small variance in GENIC performance. While it may be a good thing that the scores are consistent despite randomness, its most certainly a bad thing that they are so low. Further, these results directly contradict that of the previous work on GENIC and K-Means which states that the GENIC and K-Means similarity (sum of squared errors in their case) are comparable, granted there was no statistical testing done to say one way over the other.

treatment	ties	win	loss	win-loss-at-95%
kmeans & k=1024	0	17	0	17
kmeans & k=512	0	16	1	15
kmeans & k=256	0	15	2	13
kmeans & k=128	0	14	3	11
kmeans & k=64	0	13	4	9
kmeans & k=32	0	12	5	7
kmeans & k=16	0	11	6	5
genic & k=32	4	6	7	-1
kmeans & k=4	6	4	7	-3
genic & k=1024	6	4	7	-3
genic & k=64	7	3	7	-4
genic & k=128	7	3	7	-4
genic & k=256	6	3	8	-5
genic & k=16	6	3	8	-5
genic & k=512	4	3	10	-7
genic & k=4	0	2	15	-13
kmeans & k=1	1	0	16	-16
genic & k=1	1	0	16	-16

Figure 38: Mann-Whitney U tests for intra-cluster similarity of GENIC and K-Means. GENIC does overwhelmingly bad on on all accounts with none of its treatments having a positive record.

6.4 Implications of Results

Looking at the performance of K-Means as k in increased shows a direct correlation between the number of clusters generated and intra-cluster similarity. While this could mean that we should focus our efforts on generating cluster-sets with many clusters, there is also the trade-off with respect to time. GENIC was an attempt at leveraging the trade-offs with no avail. Given the similarity performance of GENIC, it has been shown to be not acceptable for our tasks at hand.

More importantly however, when asking what a good value for *k* is, is what the data tells us. How many clusters inherently exist in the data? The basic mathematics behind similarity will usually yield sets with many clusters a higher similarity score than those with less. Imagine if you were to measure the differences between every person in the US as lumped into one massive group, you would not find much similarity (the case when k = 1). Now if you were to continually arbitrarily split that group into sub-groups, you would eventually have a small enough population that similarities could begin to rise. Setting a static value for *k* for all datasets is not the answer. Each dataset is unique. They each have different sizes, number of variables, variable types, noise, etc.; Given this, what is important is to determine an appropriate value for *k* for your dataset. While it may be more time efficient to produce fewer clusters and more accurate to produce more, the best bet is to find the closest approximation to the inherent rank (number of clusters) of the dataset, and work from there.

6.5 Future Work

This experiment has shown us that GENIC cannot suit our needs, and that while K-Means may do a good job, it will take a long time to get there, which leads us to look for other alternatives. In our last report we mentioned an optimization step for K-Means called canopy clustering, future work will further explore evaluate this method in the same manner that GENIC was evaluated.

Other areas for improvement are the data collections we have to work with. We have in our possession, 2-3 additional text mining dataset that we have yet to fully evaluate due to their size and the computing time needed to run the numerous trials. The additional benefit of these new datasets is that they are, unlike the STEP/EXPRESS datasets, supervised, meaning each document has been pre-classified with a label that can later be used to assess clusters using external validity measures. Adding more, non-similarity based measurement criteria for clustering and classification as well is another direction we are headed towards. While cluster similarity is a useful insight into the clusters, a more practical measurement would better suit our needs. Currently in the works is a method for evaluating how well our algorithms perform within HAMLET, as used by a user.

7 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 39.

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. Point 3 (user studies) is the focus of the next few months. We know have all the support code in place for that work. Now we can turn to user trials.
3. We have a design for parsing multi-documents, which will be implemented next quarter.

The big push for the next quarter will be testing the Rochio user feedback loop. While the implementation works, and all the GUI support is in place, the open question will be "for technical users working on archival documents, will the feedback cycle offered by Rocchio add (any) value to the HAMLET system.

For each data set in the Rocchio trial, a certain number of queries will be prepared. In general, these queries will be taken from the text of bug reports. Users will be given a random data set and query to test. They will interact with HAMLET over multiple cycles of the user feedback loop, taking notes about the results returned. Although the

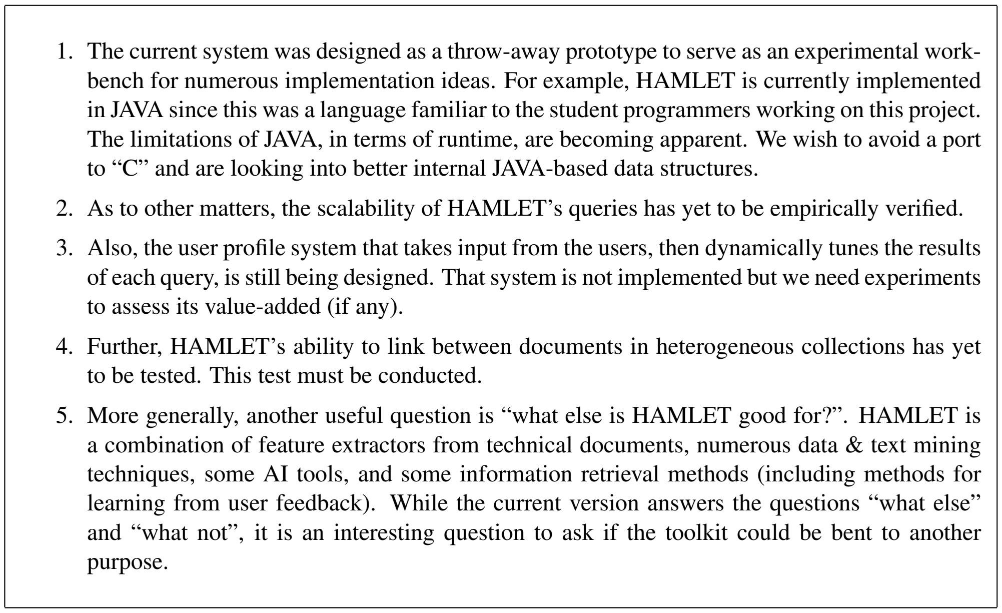

1. The current system was designed as a throw-away prototype to serve as an experimental workbench for numerous implementation ideas. For example, HAMLET is currently implemented in JAVA since this was a language familiar to the student programmers working on this project. The limitations of JAVA, in 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 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. That system is not implemented but we need experiments to assess its value-added (if any).
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 39: Our current todo list.

queries and data sets are controlled, the individual like/dislike decisions made by the user will be unique and will depend on their level of domain knowledge.

The purpose of user trials is to test the usefullness of the feedback mechanisms. These trials mirror real-life use of the HAMLET software, and the data recorded by these users should yield interesting results. Some of this data could include the number of new documents returned, the number of rated documents that remain in the list, and how many of those are liked or disliked. Knowing how many rounds of feedback that a user has to go through to find the class they are looking for should provide a good starting place for the future improvement of these mechanisms.

Because of the recursive nature of the Rochio formula, fewer new results will be found as more items have been rated. Over time, the point in space that your query represents will move less and less. Eventually, no new results will be returned. To escape this trap, Viapianni et al [**?**] have found it useful to always replace a certain percentage of your results with ”suggestions.” These suggestions should be similar items, but different enough that they would not normally be returned with the nearest neighbors. They propose several different ways to generate these suggestions. Any of the methods that they suggest could be implemented in the HAMLET system.

Our current plans call for a local user trial using university students. JAVA will be the technical document of choice because every student in the Computer Science program will have a certain level of expertise with the language. Another reason to favor JAVA is that there are a large number of open-source JAVA projects that could be used for data sets. Our current JAVA data set is based on the HAMLET code base, but future sets could include data from the Eclipse project or the jEdit program.

Note that if STEP case studies, with real-live users, come to hand, we would move on to trials with STEP (once we have debugged our experimental method using JAVA).

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

[5] 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.

[6] 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`.

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

[8] 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.

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

[10] 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.

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

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

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

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

The HAMLET PROJECT: status report

Changes to this document since last submission

- §2.2 has a clearer overview of the HAMLET system (e.g. see Figure 3);
- §4.1, §4.2, §4.3 contains our our new screens, including all those required for the user feedback system. The support text for all those figures have been changed.
- Most of §5.4 has been changed and better text has been added describing GENIC (§5.4.1) and K-means (§5.4.2).
- The connection of HAMLET to the Rocchio algorithm is better explained in the new text of §5.8.
- Chapter 6 is entirely new and describes our GENIC vs K-means shoot-em-up.

Work performed since last submission

Since our last report, our effort has gone into two tasks. Firstly, as shown in Chapter 6, we spend much time running experiments assessming different clustering algorithms. Secondly, we had to prepare the system for the user feedback triaks planned for 2009. This required:

1. recoding the interface to handle preference information (see figures 11,12,13,14)
2. changing much of the internal structure of HAMLET to take into account user profiles for mulitple users across multiple sessions.

Task 2 was a surprisingly tedious task: HAMLET's original design assumed a single user and that very pervausive throught the entire code base. Much work was required to recode all the internal structures to handle multiple users running multiple sessions.

Resources used since last submission

Travel: No travel costs were incurred as part of this work for this quarter

Software: No software purchases.

Hardware: This work was performed on three Dell Laptops, (bought May 2008) and one Apple Mac (June 2008).

PersonneL: The work for this quarter was performed by:

- Tim Menzies, associate professor LCSEE
- Andrew Matheny, Masters student, LCSEE
- Gregory Gay, Masters student, LCSEE
- Adam Nelson, undergraduate programmer, LCSEE