

Query by Browsing using Genetic Algorithms

Stephen Wattam

Lancaster University
Lancaster, LA1 4WA
+44 1524 510 319

s.wattam@lancaster.ac.uk

Abstract

Recently much work has been put into the idea of creating database queries in a data oriented manner; omitting overtly technical user interface requirements and allowing the creation of highly complex selection rules without specific, in-depth knowledge of the desired result. Almost all of these efforts have relied on conventional deterministic categorisation algorithms to build their rule sets by analysis of the selection only, creating accurate queries without concern for any other desired properties. This paper investigates the use of non-deterministic machine learning techniques in the creation of queries that conform to conflicting or logically orthogonal desired properties.

Categories and Subject Descriptors H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval

General Terms Human Factors, Design, Experimentation

Keywords Genetic algorithm, Machine learning, Data retrieval, SQL, Query strings

1. Introduction

The task of meaningful data extraction from large relational databases, especially when extracting automatically-garnered data, has proven to be one that consistently stretches conventional descriptive query techniques. Users, when describing a query, must themselves select and resolve patterns in data to provide their desired result, in many cases selecting sub-patterns which they wish to exclude from the selection.

Whilst this method of query creation is exceptionally powerful it is useful only to those who understand many varied and, usually, hard to acquire pieces of information regarding the system they are using: an effective query built in SQL will often involve having a knowledge of many tables, their relationships and how the data is used once extracted.

This paper builds on the principles of Query By Browsing (2), allowing data extraction where the pattern used to select data is secondary to the data to be selected: a user may have little or no idea of what links the data beyond the principle that they desire it for a set task, which may contain an arbitrarily complex set of requirements. Although ultimately user-focused it is primarily an

exploration of technologies that could complement the interface chosen

Traditional methods such as building decision trees from significant traits assume that only one property of the returned data is of importance to the end user: accuracy. Whereas this most accurately mimics the way traditional queries work it is of questionable use to the end user: the cleanliness of the SQL string, execution time of the query or ignorance of certain database properties may also be of great importance depending on the application for which the query is being constructed.

The use of evaluated machine-learning techniques in this field allows for such arbitrary optimisations to occur, reducing the propensity for a query to be something that will provide a correct answer at the expense of a useful one.

This paper will investigate the benefits that may be drawn from the use of a non-deterministic genetic algorithm in order to estimate the desired trends in a given user's selection in the context both of direct user interaction and where the search technology may be hidden behind a less direct user interface.

This paper begins with an overview of the technologies and ideas used within. This is followed by an in-depth review of the prototype implementation and its flaws and results. This is followed by a review of similar work in the field and finally a review of success and failure relative to the original goal: the conclusion.

2. Background

2.1 Ostensive selection

The underlying principle of Query By Browsing is that of changing the query building process into its ostensive (or, in specialist cases, extensive) form. Queries constructed using this method are, however, expressed in an intensive manner, as they must be in order to be processed.

The construction of intensive queries from a finite subset of possible selections is ordinarily accomplished by the user. This may be direct: 'I wish for all those entries over 4 months old' or derived from a subset: 'It would appear that all users of version 3.4 require a patch ergo I wish for all those entries with a version number of 3.4'. In the case of indirect construction the conversion to intensive form is often non-trivial and includes features of data of which the end user is unaware.

To illustrate this point let us examine a common method of traversing the web. Since search engines have only the same amount of data available to them as the user does (discounting directly calculated metadata) it is necessary for a user to derive intensive definitions of the data they wish to extract. It is common for those who use search engines frequently to learn small excerpts such as 'index of' or 'generated by Apache' in order to search for pages of automatically generated descent, or to use filename regular expressions to weight certain websites' content.

Systems such as 'StumbleUpon' (9) provide a more ostensive selection method, at the expense of immediate availability: sites similar to those one has visited are presented, based upon the relative popularity of other users possessing similar interests. This does not easily allow for quick selection of a desired topic, however, browsing their categorisations allows for quick recall of sites which are statistically likely to be desired. This is similar in mechanism to website advertisement placement, which attempts to extrapolate desired results based on at most a few data points (4) (5).

2.2 Improvements over Query by Browsing

The current implementation of Query by Browsing builds its intensive queries by deterministic analysis of the given data: more specifically it uses an adaptation of ID3 algorithm (8) to identify common points, building a decision tree that is then resolved into an SQL string.

This provides a highly accurate and reliable mechanism of selecting the sample data, however, this is arguably its greatest failing. Users undoubtedly want accuracy in their selections, however, the existing method of building query strings takes into account only accuracy of selection, and only accuracy of selection within a sample subset. Depending on application users are likely to desire other concerns such as:

- Cleanliness, length, ease of comprehension of produced SQL
- SQL execution speed
- Over/under sampling
- Suitability for transference to other data sets (even other table structures)
- Field, structure or data specific selection
- Pattern selections (systematic sampling)
- Fixed size returns

There are a number of obvious examples for each of these, many based on different incarnations of a simple search: a review site, for example, will desire oversampling with reference to some product-specific fields and under sampling on more generic properties that effect selection of products in which the user is uninterested. Currently QbB does not allow for any such selections.

2.3 Sample data selection

In such a situation that an algorithm must identify traits in selected data there arises the problem of what sample to provide. Technically this sample must embody all possible variations across all fields, whilst retaining multiple examples of each trait. This is impossibility without providing the whole dataset to the user (a possibility if the resultant query is to be run elsewhere, similar to exhaustive supervised training).

3. Genetic algorithms as a solution

3.1 The concept

The use of non-deterministic machine learning techniques allows for optimisation of problems involving arbitrary, conflicting concerns. Many non-deterministic search techniques have proven themselves too unreliable, exhaustive (and thus prohibitively slow) or susceptible to getting trapped in local maxima, however.

I am to be using a genetic algorithm to search for solutions to the problem of finding query strings that conform to a series of metrics as defined by the user's input. In order to better evaluate the non-deterministic aspects of the solution I shall not be studying the user's iteration with the interface itself; rather the nature and appropriateness of the returned string.

Through whatever interface is given to a Query by Browsing system selections are made on a provided sample of data. This is then processed by building and refining rule sets using a genetic algorithm until such time as a fitness function is met, whereby the resultant SQL is to be returned. This mechanism is similar to 'traditional' QbB, except for the manner in which rule processing is accomplished.

Within the genetic system a series of chromosomes are generated and run through a series of fitness functions. These fitness functions score the chromosome based on a series of set criteria. Throughout a series of iterations the population of chromosomes is manipulated in a manner causing similar evolution to that used in selective breeding - chromosomes (generally those with better scores) are combined during crossover and their offspring optionally mutated. This offspring is then placed back into the population, normally replacing either a weaker chromosome or one of the parents. The process is halted when a certain condition is met - generally when the best solution exceeds a set bound.

During this process certain properties make themselves apparent in the form of phenotypes: the characteristics generated by a genotype - an internal representation of properties that form a single logical function.

3.2 Concerns regarding implementation

Within a genetic system there are certain design concerns that must be addressed so as to produce not only an effective system, but also an efficient and flexible enough to resolve multiple concerns into a query within a reasonable time. Being a research system my implementation is far from optimised for speed, yet remains fast enough to produce results within a few seconds.

An iterative execution path as proposed by Marek Obitko (6) was chosen to be the basis of the genetic algorithm:

- I. Population generation;
- II. selection;
- III. crossover;
- IV. mutation;
- V. acceptance;
- VI. testing (evaluation).

This algorithm is followed during the main execution loop of the expression builder, and returns only minor statistical data on the nature of the population during execution.

Resolution of possible comparable concerns is handled by an engine that resolves retrieved data into a form where it is stored along with appropriate metadata: location within tables and type are stored within each 'token'. These tokens form the atomic building blocks of query expressions.

3.3 Design concerns

3.3.1 Encoding

The method used to encode a query is one which is likely to influence the quality of results, speed of optimisation and development of new phenotypes greatly. For reasons of statistical equivalence I represented each chromosome as a binary decision tree, as in the modified ID3 algorithm used in QbB (2).

This decision tree is composed of a series of boolean expressions relating to possible atomic queries upon the given table. Data is selected from the table in order to generate these. Leaf nodes define a desirable (true) or undesirable (false) path through the tree.

The choice of binary decision trees allows for equal weighting of concerns relative to one another, as well as the construction of wildcard based queries without dramatic skewing of the probability of selecting those rules.

Simplification of the decision tree is possible by way of recursive resolution of leaf nodes' values. As such each decision node obtains a value regarding its desirability: true if both child nodes are desirable, false if both are undesirable and 'unknown' in the event that both children resolve to a different value.

3.3.2 Crossover

An obvious crossover method for the chosen encoding is tree splicing. This allows for easy maintenance of favourable groups of selections whilst disrupting a selection enough to prove useful to the progression of the population towards a higher fitness. This method is most similar to single point crossover in linearly described chromosomes.

A second appropriate method of crossover between two chromosomes would be the selection and exchange of rules within the decision tree. This method is more likely to disrupt any grouping of genotypes, resulting in less predictable change in the final selection. This is generally considered undesirable as maintenance of phenotypes is an important part of the optimisation process.

3.3.3 Mutation

Mutation of offspring chromosomes is accomplished using any one of three methods, each of which has significantly different effects upon the process of optimisation:

- Random inversion of the desired 'true' or 'false' states of the final leaf nodes. This can have the effect of dramatically changing the final appearance of the query and so adjusts the course of optimisation in an equally extreme manner.
- Simplification of the decision tree into only those nodes that are currently being resolved into the final SQL string. This effectively 'prunes' the tree, changing the nature of its subsequent crossover or mutation operation outcomes yet retaining its current value.
- Wildcard insertion into String fields. This method is best suited to sampling of large data sets with similar data strings where wildcard hits are going to select similar data items.

3.3.4 SQL Resolution

Resolving the decision tree into a meaningful SQL query is accomplished by way of a depth-first recursive algorithm. This generates relatively ugly, hard to follow SQL due to a lack of bracket resolution, although this was not deemed to be important to the task of evaluating the effectiveness of the final query.

Resolution of decision trees may be limited to those with 'unknown' states without loss of accuracy, as any further resolution would result in the equivalent of a static comparison and hence would return all or no records, both of which are useless results in the context of data sampling. It should be noted that when resolved all trees retain their hidden complexity, as mutation and crossover is likely to transform a branch into one of 'unknown' state. In samples where the desired return is a superlative number of records queries with superlative selections are likely to succeed: it is not uncommon for a '...WHERE 1' or '...WHERE 0' to result in these instances.

3.3.5 Selection

Steady-state selection is used, with chromosomes replacing the weaker ones in the population. The ability to vary the selection sample allows for the whole population to be selected if necessary.

3.3.6 Acceptance

Certain chromosomes may be reliably and definitely excluded from the population by the use of a *Sanitiser*, which is able to reject

chromosomes as being unsuitable for acceptance back into the population. This is generally done to ensure that conditions such as comparisons between two literals never propagate through the population.

3.3.7 Initialisation

Initially all decision trees are built using a statistically weighted recursive algorithm that selects tokens from the given data set and assembles them into expressions. Those that are malformed are rejected. Exactly what constitutes malformed is a matter of opinion here: invalid or unwise comparisons such as those involving anything other than a field name and a literal of its requisite type may be considered unacceptable or may cause a reduction in fitness for that chromosome.

The depth and structure of these initial trees appears to make little difference to the outgoing query, due to the fact that in a given population enough good examples exist in order to overwhelm the unsuccessful forms.

3.3.8 Fitness and evaluation

Evaluating the fitness of a chromosome is one of the most influential aspects of the genetic system. Fitness functions are evaluated separately to be numbers between arbitrary boundaries, as defined within themselves. This fitness value may be combined with other functions according to an arbitrary weighting to provide a final fitness value for a given chromosome.

Readability, for example, is easily affected by the shape of the tree: trees that consistently evaluate on their 'High' branch will create queries with long strings of 'AND' keywords; those with consistent strings of 'low' evaluations will resolve to SQL strings with many 'OR' options. Both of these are preferable to complex back-and-forth mixes that produce hard-to-follow logic.

In order to retain core functionality of the system the primary method of ranking a population should be selection of data as desired by the user. This fitness function takes into account the data in each record, hashing real selections and making comparisons weighted by a user's selection weight. Those records that have positive weight will reward those chromosomes that selected them; those with negative weightings will reduce the chromosome's fitness value. Due to the restricted nature of the record-matching this fitness function effectively limits the execution of the SQL generation algorithm to the same data set from which the user made selections.

This system effectively mimics the QbB 'yes, no, do not care' system by providing three definite values: -1, 0 and 1 ranging from undesirable to desirable respectively (incidentally it is possible to skew the results of the fitness function dramatically by applying extreme values to these fields). This method allows for greater selection resolution as one may rank selections in order of preference (a task likely to be carried out by frequency of use or some other automated mechanism in a final system).

Although this remains the primary method of ranking chromosomes there are others which may be appropriately applied. The brevity of a returned SQL string is often of concern, as may the balanced nature of the tree (balanced trees being likely to select as much as they actively reject) or the 'portability' of the final query (a metric likely to be evaluated using other databases or relating to the number of wildcarded or ranged searches featured).

Continuous and discrete data types must have their comparison operators weighted in order to create meaningful comparisons: there is little sense comparing discrete data with the greater than operator, for example. Because of the unpredictable correlation of field type to data continuity this inclination is implemented as a fitness function; the propensity of meaningless operator combinations to occur is thus lowered but not entirely removed.

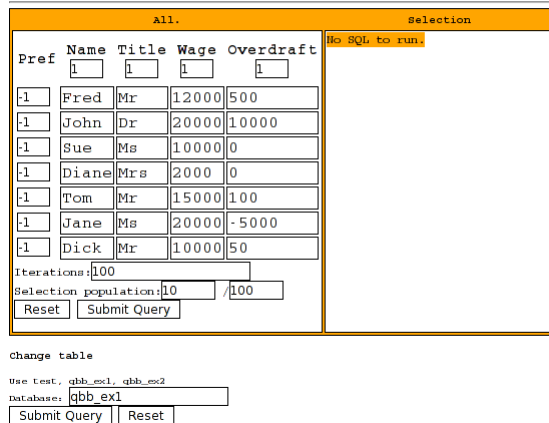


Figure 1. The web interface, with no results

4. Prototype and results

4.1 Prototype overview

A prototype fitting the design pattern above has been created in the form of a standalone Java system interfacing directly via the MySQL connector/J to a MySQL database. A PHP interface was created to provide a more intuitive interface.

This interface was designed as a technical development platform and was not intended to present a user-friendly appearance.

As can be seen from Figure 4.1, a form awaits the user below an existing table selection, from which they may select SQL to run in order to generate a selection. This version of the interface assumes that users wish to use all data in a given table to generate SQL strings, and the provided tables are deliberately of appropriate size (around 10-20 records) to allow this. Data in the examples is taken from the QbB web distribution in order to perform a more accurate comparison with the same sample characteristics.

The darker a line is the greater priority its selection is afforded. This provides an easy method of recognition when compared to the 'output' table, which appears in the same format: lines that have been selected by the resultant SQL are presented in black.

Upon selection of a table to use the relative desired fitness values are keyed into the text boxes and the algorithm run using the 'submit query' button. The end condition used for the algorithm is an iterative one, allowing the algorithm to commit a set number of iterations before termination. The number of chromosomes selected for crossover and the population size is also editable.

Since the purpose of this paper is to examine the nature of genetic algorithms rather than specific properties of a given implementation I shall be using a variety of fitness functions designed to apply to each given solution. An overview of the fitness functions, selection methods and sanitisers are provided in appendix C.

4.2 Rule decision tree representation

All possible tokens within the system are stored along with metadata defining their field name, types and information regarding resolution into a valid string representation (for use in a SQL query). These tokens are then assembled into rules which form the basis of decision tree nodes (Figure 4.2). The nodes are then constructed into a decision tree which ends in leaves defining a selection (true) or a rejection (false). Whilst it is possible to construct arbitrarily complex syntactically accurate rules in this manner the prototype restricts these to single pairs. This ensures that all crossover and

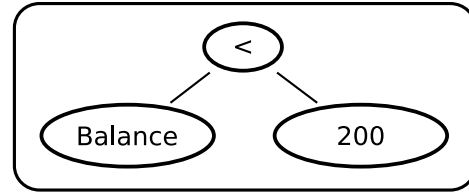


Figure 2. A single decision tree node

mutation relating to tree structure need only be applied to the decision tree rather than the structure of the rule tree.

The tokens used in rule generation are, in this prototype, generated using database data only. It is conceivable that an extension to this prototype would insert wildcards into string fields, select random values from enumerated fields and modify numbers at the token generation stage. This system relies on mutation to perform those (and other) tasks, practically reducing the number of such features.

Rules are resolved such that their evaluation leads to another rule or a leaf node determining whether the given possible route is to be returned or not. The evaluation of a rule to be true causes the resolution algorithm to follow the 'high' child node, represented by way of being the leftmost branch in Figure 4.2, whereas a false evaluation causes the low (right) side to be followed. In the

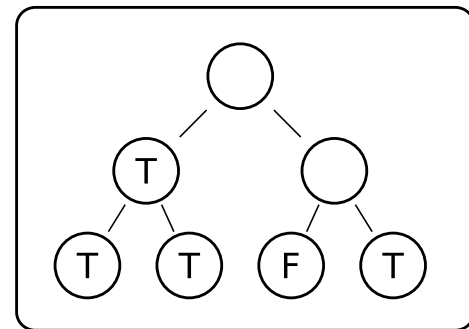


Figure 3. The process of resolving rules

prototype this is handled by a depth-first recursive algorithm that discards unnecessary rules in order to build query in a bottom-up manner from the lowest known 'true' or 'false' state.

4.3 Undesirable effects

The method of randomly generating collections of chromosomes can result in some common errors. Although it is generally possible to remove them there are cases in which such selections may help the final result.

4.3.1 Nonsensical output

It is clear from certain selections that the current fitness functions do not constrain enough: a selection of both John and Tom using the table 'qbb_ex1' has been known to return incomprehensible and perplexing queries such as:

```
SELECT * FROM 'test'.'qbb_ex1' WHERE
  NOT 'Overdraft' >= 10000 AND
    (NOT 'Name' <= 'Jane' AND
      ('Mr' = 'Title' AND
        (2000 <= 'Wage')))) OR
  'Overdraft' >= 10000 AND 'Overdraft' > 0
```

Whilst this selects the correct range it is clear from the output that it satisfies the selection accuracy fitness function to any great degree.

Comparisons such as

```
'Name' <= 'Jane'
```

are clearly nonsensical, or, if useful, can be guaranteed to apply accurately only to the finite data set selected and thus are of little value. There are functions to prevent this occurring in the prototype.

4.3.2 Duplicate statements

Another common flaw in the generation of query strings is the duplication of rules such that they form a tautology or contraction. This is relatively difficult to resolve due to a lack of intelligence in the query building process, and is likely to be best resolved during SQL resolution itself (as, once more, part of the logic may be altered during crossover or mutation, causing a meaningful result). Such an example can be found below:

```
SELECT * FROM 'test'.'qbb_ex1' WHERE
NOT 'Name' >= 'Sue' AND 'Name' >= 'Title' OR
('Name' >= 'Sue' AND 10000 < 'Wage')
```

Here we can see that the tautology in question is nested within a comparison with two other properties, being those that form the desired selection. This can best be solved either by informed resolution of logical concerns at a post-processing stage or introduction of further SQL-cleanliness fitness functions, weighted accordingly. There are currently no functions to prevent this occurrence in the prototype, as they do not influence resultant selection and are easily picked out by hand.

4.3.3 Logical redundancy

A form of undesirable query that is more difficult to identify is that of rules with overlapping selection ranges causing redundancy. Whilst this may be desirable in certain cases (selection of names, for example) it is largely undesirable in continuous and numeric fields. It is easier to resolve in a purely numeric form, which is made more likely to occur by use of fitness functions that reward meaningful comparisons.

```
... 'Overdraft' >= 10000 AND 'Overdraft' > 0
```

This example clearly shows such a selection, however, without in-depth analysis of the nature of each field relative to rule combinations this is hard to accurately and reliably resolve. Once more this form of fallacy is the comparison of fields with numbers that are beyond the range of any data which resides in that field.

4.4 A simple test

Query by Browsing, GA style. X

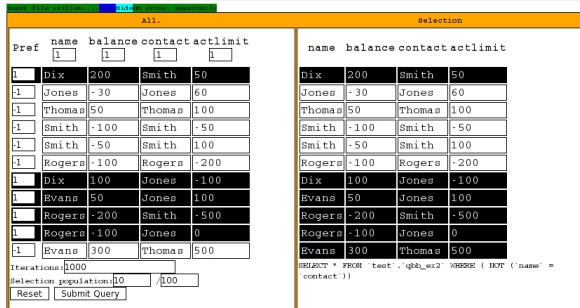


Figure 4. A query using table qbb_ex2

The test illustrated in Figure 4.4 was run on the second QbB sample table 'qbb_ex2'. Please see appendix A for a full detail of its contents.

The SQL returned from this query was:

```
SELECT * FROM 'test'.'qbb_ex2' WHERE
NOT 'name' = 'contact'
```

This appears to be relatively accurate. The deterministic QbB algorithm determines the following:

```
SELECT * FROM qbb_ex1 WHERE
Overdraft >= 100 AND
Overdraft != 500
```

This is clearly more accurate, but is arguably of lesser portability: a user is unlikely to have deselected that one exception to the rule based on that field alone. It would be accurate to say, however, that the former is an example of clear and applicable SQL - one of the fitness functions valued during rule generation.

It is, perhaps, a failing of both algorithms that the 'exception to the rule' record is analysed on non-specific fields, and this presents a new problem: how does one favour generic fields for generic selections and yet use specific contradictions on those which are likely to be intentionally omitted.

4.5 Priority based selection

Query by Browsing, GA style. X

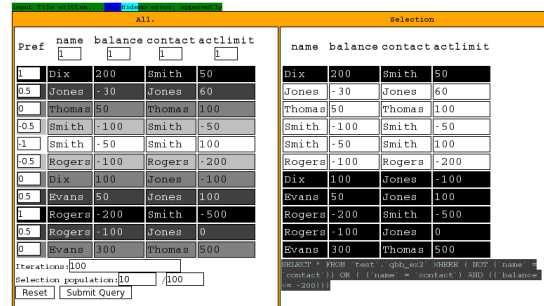


Figure 5. A query using table qbb_ex2

Figure 4.5 provides a good illustration of the bias afforded by sample properties. The SQL returned from this sample is as below:

```
SELECT * FROM 'test'.'qbb_ex2' WHERE
NOT 'name' = 'contact' OR
('name' = 'contact' AND
'balance' <= -200)
```

Query by Browsing, GA style. X

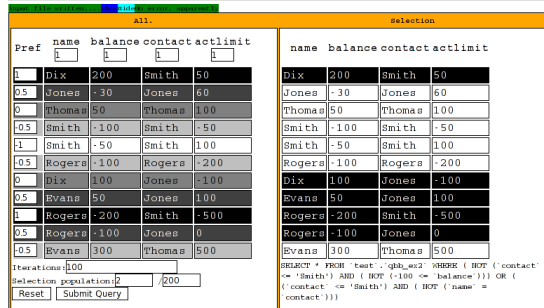


Figure 6. A query using table qbb_ex2, emphasis on SQL accuracy

Reduction in the strength of the simplification and comparison sanity fitness function results in a more accurate if largely indecipherable query, as illustrated in Figure 4.5:

```
SELECT * FROM 'test'. 'qbb_ex2' WHERE
(NOT 'contact' <= 'Smith' AND
 NOT -100 <= 'balance') OR
('contact' <= 'Smith' AND
 NOT 'name' = 'contact')
```

Ignoring the redundancy it is clear to see that the fitness functions have steered the genetic selection towards a situation where the ultimate result is defined not only by accuracy but also brevity (there are 'better' selections when evaluating only user choice, however, the rules which select them are complex).

A question arises in instances where a prioritised selection is made as to what the user was really intending to select: one could assume several sets of desired data, each with differing importance, or a more flexible mix of competing selections.

5. Evaluation of performance

5.1 Samples

The table 'qbb_ex1' appears to present the non-deterministic algorithm with greater categorisation challenges than does 'qbb_ex2'. This would appear to be due to many factors:

- The relatively small number of examples (7 as opposed to 11), causing generally high levels of deviation between individual records;
- the relatively low number of duplicate entries in many fields;
- the weak relationships between fields in the same table.

It would appear that table 'qbb_ex2' plays to a genetic implementation's strengths in that it contains many duplicate data points, making random rule generation probabilistically more likely to succeed first time and raising the likelihood that a mutation will improve the chromosome.

The greater number of records also allows for a larger vocabulary and reduced granularity in the fitness functions, allowing positive phenotypes to persist in states that are sub-optimal yet do not excel (the probability that a non-favourable edit will cast a chromosome to the bottom of the ranking is reduced).

5.2 Flexibility

Given the varied nature of results returned it is clear that a genetic approach is capable of generating many different forms of query, each with very different attributes. In this manner it sets itself aside from more deterministic rule based methods in that the concerns applied to the end result vary it dramatically and unreliably.

The question though is perhaps more accurately described as 'are the given results useful'. That is so domain specific as to be unanswerable in the general case, however, the nature of returned results has indicated that the use of genetic algorithms in rule-building is capable of providing queries that best fit a general rule rather than its specific intricacies.

5.3 Accuracy to intended result

Ultimately the success of the prototype must be judged with reference to the degree to which results matched the intended. This assumes that the algorithm thus attached all appropriate weights to each individual component of the query, making its intensive definition as rich as the ostensive one provided.

This is particularly hard to judge partially because of the many subtle metrics humans apply when analysing data. In matching complex selections with the need for 'compact' SQL strings the

genetic approach has proven itself capable of resolving orthogonal concerns. To what degree these concerns may be applied before the mechanism breaks down is a matter for further research.

Largely these 'extra' concerns are necessary in order to build query strings that match up to the quality of one built by a well-informed algorithm such as the method used in QbB.

In contexts where a priority (either that of record or field selections) is required an algorithm must decide not only on the relative importance of various query features (which involves implicit evaluation of how much each rule affects the results) but also which properties of each individual concern are required. This is accomplished more easily by something such as a genetic solution which is capable of running effectively whilst still remaining largely ignorant of data selection. This is where comparisons with conventional techniques become favourable: QbB, for example, is incapable of trading accuracy for any improvement, no matter how great the other improvement or how small the reduction in query accuracy.

It is this trade-off capability that provides most power: a well-balanced selection of fitness functions ought to be capable of mimicking the judgement of a human, at least for a majority of selections. This requires, however, both carefully-designed fitness metrics (query simplicity is, for example, of less use than query readability in most contexts) and well-balanced, targeted application of them to data (well-chosen samples).

6. Related work

A similar theme of study appears to have been followed by Jong-Min Park (7) in a study of conversion of textual data from web browsing (through keyword and topic detection) into a list of topic boundaries. These boundaries allow the construction of queries that can retrieve data from within, or excluding, a single topic. This would seem to be a suitable system to act as an informed input device to an algorithm such as the one discussed in this paper, as the results would then benefit both from a highly accurate targeted approach and a human categorisation and comprehensibility.

On a related tangent to the retrieval of meaningful data is research into the restriction of returned result sets to a predictable size by way of matching data to topics (3). This technique could easily be applied in order to 'crop' erroneous points from a largely correct selected set, and would, in its native form, complement methods of 'fuzzy' selection outlined in this report.

7. Conclusions

The prototype successfully demonstrates some aspects of improvement over traditional query building techniques, whilst also displaying a number of weaknesses that are attributable primarily to errors of implementation.

The greatest challenge of this field is recognition of concerns which are important to the user and construction of intensive rules which are capable of returning data in a manner which conforms to them. My prototype indicates that genetic algorithms are capable of this to an extent that, in my system, would seem to be limited by the nature of its fitness functions. The restriction of problem domain would provide one method of simplifying the requirements of the user enough to create tailored fitness functions, as might the production of many 'selectable' concerns, allowing the user to manually choose importance.

It would seem that the clearest manner in which to present these options is to state that data selection creates the rules and that fitness function selection creates the properties which those rules possess. This would be a useful paradigm to adjust an already specialised form of the algorithm using purely statistical means (such as the gradual improvement of search results as a user uses a website),

although it does little to simplify the creation of queries as initially stated.

Another form of specialisation that may prove useful is not task-oriented but data-oriented: a system which can rely on certain headings or encodings within a table is able to specifically generate or manipulate them in a manner which will prove useful to the final query. This is essentially an extension of the token and rule generation stage of the existing algorithm.

There is clearly a lot of variation caused by the data selection from which the user may choose: algorithms that find patterns (or a lack thereof) within a set are ideal for extracting subsets for this application, and the parameters for such require further study.

References

- [1] A. Dix. Query by browsing on the web. <http://www.meandeviation.com/qbb/qbb.php>, 01:01, Accessed 20th March 2008.
- [2] A. Dix and A. Patrick. Query by browsing. *Proceedings of IDS'94: The 2nd International Workshop on User Interfaces to Databases*, pages 236–248, 1994.
- [3] R. Godin, C. Pichet, and J. Gecsei. Design of a browsing interface for information retrieval. *SIGIR Forum*, 23(SI):32–39, 1989.
- [4] Google. Adsense tour. http://www.google.com/services/adsense_tour/page5.html, 01:01, Accessed 19th March 2008.
- [5] D. Inc. Dart for advertisers. <http://www.doubleclick.com/products/dfa/index.aspx>, 01:01, Accessed 19th March 2008.
- [6] M. Obitko. Introduction to genetic algorithms. <http://www.obitko.com/tutorials/genetic-algorithms/>, IV:Outline of the Basic Genetic Algorithm, Accessed 18th March 2008.
- [7] J. Park. Intelligent query and browsing information retrieval (iqbir) agent, 1998.
- [8] J. R. Quinlan. Induction of decision trees. *Machine learning*, 01:01, 1986.
- [9] StumbleUpon. The stumbleupon website, about page. <http://www.stumbleupon.com/about.html>, 01:01, Accessed 18th March 2008.

A. Tables (1)

Field name	type
Name	varchar(20)
Title	varchar(4)
Wage	int(11)
Overdraft	int(11)

Table 1. Structure of table qbb_ex1

Name	Title	Wage	Overdraft
Fred	Mr	12000	500
John	Dr	20000	10000
Sue	Ms	10000	0
Diane	Mrs	2000	0
Tom	Mr	15000	100
Jane	Ms	20000	-5000
Dick	Mr	10000	50

Table 2. Content of table qbb_ex1

Field name	Type
name	varchar(15)
balance	int(11)
contact	varchar(15)
actlimit	int(11)

Table 3. Structure of table qbb_ex2

name	balance	contact	actlimit
Dix	200	Smith	50
Jones	-30	Jones	60
Thomas	50	Thomas	100
Smith	-100	Smith	-50
Smith	-50	Smith	100
Rogers	-100	Rogers	-200
Dix	100	Jones	-100
Evans	50	Jones	100
Rogers	-200	Smith	-500
Rogers	-100	Jones	0
Evans	300	Thomas	500

Table 4. Content of table qbb_ex2

B. Parameters of the prototype

Please note that any functions designed only to modify the behaviour of others have been omitted from this list, though have been used in order to alter weightings and probabilities during execution.

B.1 Fitness functions

- Rule counting - attempts to determine rule complexity from the number of active rules in a decision tree.
- SQL Length - resolves SQL and then ranks it according to deviation from a set number of terms.
- Meaningful comparisons - Favours those functions that make comparisons with 'meaningful' operators. Similar to the meaningful comparison sanitiser only less harsh
- User choice - uses the user's choices of fields and records in order to rank chromosomes.
- Random - little used but capable of applying a level of entropy.

B.2 Sanitiser methods

- Meaningful comparisons - rejects any chromosomes that use rarely-applicable operators (i.e. greater/less than).
- Type comparison - rejects any chromosomes that compare rarely-compared types, such as a String and an Integer.
- Logic resolving - resolves the decision tree's logic and rejects any chromosomes which resolve to be exclusively true or false.

B.3 Mutation method

- Leaf editing - randomly inverts the leaves of a decision tree.
- Wildcard insertion - inserts wildcard characters into String fields.
- Tree simplification - resolves a decision tree into its simplest form without changing its final SQL (alters the behaviour of subsequent edits of the tree).

B.4 Crossover

- Tree splicing - selects a point on both decision trees and joins one subtree onto the other, creating a new decision tree.

B.5 Selection

- Roulette wheel selection - performs selection with a chance of selection being $(\text{chromosome fitness}) / (\text{population fitness sum})$.
- Random selection - selects a chromosome without consideration of the fitness value.