

Refiner++: A Knowledge Acquisition and Refinement Tool

Andy Aiken

Computing Science Department,
University of Aberdeen,
Aberdeen, Scotland, UK
aaiken@csd.abdn.ac.uk

Derek Sleeman

Computing Science Department,
University of Aberdeen,
Aberdeen, Scotland, UK
sleeman@csd.abdn.ac.uk

ABSTRACT

Refiner+ is an algorithm that detects inconsistencies in a set of examples (cases) and suggests ways in which these inconsistencies might be removed. The domain expert is required to specify which category each case belongs to. Refiner+ infers a description for each of the categories and reports any inconsistencies that exist in the dataset. An inconsistency is when a case matches a category other than the one in which the expert has classified it.

Refiner++ is a new Java implementation of the (LISP-based) Refiner+ algorithm. The original algorithm was modified in several respects. Refiner++ was designed to be an application that would allow a domain expert to enter data, manipulate it and examine it without the need for a knowledge engineer to act as an intermediary. At the time of writing, the Refiner++ system has been presented to three experts to use on problems in their domains: anaesthetics, educational psychology, and intensive care.

The work is ongoing, but at this stage we have identified types of tasks Refiner++ is particularly good at solving, and some for which it is less useful. The application has met with great interest from domain experts, but as yet none seem willing to use it directly. Enhancements to the UI and the materials used to introduce a new domain expert to the conceptualization task inherent in a Refiner++ session are under way. Shortly we plan to run a series of new studies with further domain experts.

Categories and Subject Descriptors

I.2.1 Applications and Expert Systems – *medicine and science*.

I.2.4 Knowledge Representation Formalisms and Methods – *representations*.

I.2.6 Learning – *knowledge acquisition*.

Keywords

Knowledge acquisition, knowledge capture, knowledge

refinement, classification, case-based reasoning, Refiner, Refiner+.

INTRODUCTION

The basic philosophy of the original Refiner system [7] is that it is better to detect errors and inconsistencies in sets of cases before a CBR (Case-based Reasoning) session than during one. Refiner reads in a set of labelled cases (i.e. cases for which the domain expert had specified the category to which it belongs), background knowledge if it was available, and the system generates a generalized description for each of the categories. Moreover, this process is performed incrementally i.e. each new case causes each of the category descriptions to be recalculated. Moreover, at each stage if the case base is inconsistent (i.e., a case can be classified as belonging to more than one category), this fact is flagged and Refiner suggests a number of ways in which that latest inconsistency could be removed. The system requires the domain expert to choose the appropriate modification. The types of modifications frequently suggested include:

- Exclude value
- Change value
- Reclassify case
- Shelve case
- Add a new field

The principal application domain used with Refiner was decision making in a Hospital Hypertension out-patient clinic about when patients should next be seen. Six decisions were made: SHORT-TERM-Clinic-Appointment (Clinic-SHORT for brevity), Clinic-MEDIUM, Clinic-LONG, GP-SHORT, GP-MEDIUM & GP-LONG. As Refiner asked the domain expert to remove an inconsistency as soon as it arose, this led to changes being made at one “cycle” which would sometimes be “undone” during a later “cycle”, as the domain expert did not have a good overview of the data-set. Therefore, it was decided to re-implement the algorithm as a batch algorithm, which was named Refiner+ [8]. Refiner+ reads all the cases before it creates any of the category descriptions or detects any inconsistencies. The great advantage of this approach is that the algorithm now has an overview of the data sets and so can suggest an ordered set of changes (that is, changes which are likely to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

K-CAP'03, October 23-25, 2003, Sanibel Island, FL, USA.
Copyright 2003 ACM 1-58113-000-0/00/0000...\$5.00

remove a sizable number of inconsistencies are listed before those which only remove a single inconsistency).

The current system, Refiner++, processes cases as a batch as did Refiner+. The system is now implemented in Java (as opposed to LISP for the earlier systems); owing to the ubiquity and portability of Java, Refiner++ can be run on a laptop computer which has significantly changed the nature of its use by domain experts. Currently we are primarily investigating the use of Refiner++ to prototype sets of cases for a domain. We have thus encouraged domain experts to describe maybe two or three cases of each of the categories in a domain (say childhood infections, types of vehicles etc) and then to use Refiner++'s analytical capabilities to infer category descriptions and, if appropriate, a set of inconsistencies and suggestions as to how those inconsistencies might be removed. This feedback allows a domain expert to see if the case descriptors chosen are sufficient to discriminate between the different categories and whether the category descriptions inferred by Refiner++ are appropriate.

The rest of the paper is organized as follows:

- Refiner++ (description of the algorithm and the UI)
- Using Refiner++ (our interviews with domain experts)
- Conclusions
- Ongoing Work
- Related Work

REFINER++

The Algorithm

As noted earlier, Refiner++ is based in the Refiner+ algorithm, which in turn was based on the Refiner algorithm. Just as Refiner+ improved upon the original algorithm, Refiner++ is an incremental improvement upon Refiner+.

Phase 1a: Data entry

After having decided on a topic in which there are categories between which the domain expert wishes to discriminate, he should decide which descriptors¹ are needed to describe the cases, and their types.

Background knowledge is also added at this stage; Refiner++ uses background knowledge about the fields in the dataset to help disambiguate categories. Taxonomies are the only form of background knowledge currently used by Refiner++, but other types of background knowledge are planned.

The domain expert adds a collection of cases (examples), each of which is assigned to one or more categories. Each case should have a value for each of the fields, although Refiner++ can cope with missing data / sparse datasets.

¹ The Refiner++ application refers to descriptors as *fields*. The two terms are, for our purposes, interchangeable.

Table 1 shows a sample dataset which contains data relating to hypertension patients. Each case consists of three fields: Age and DBP are numeric fields, and Disease is a taxonomic field which uses the Disease taxonomy shown in Figure 1. The domain expert has assigned each of these cases to one of two categories, A or B. We will use this dataset to illustrate how Refiner++ performs its analysis.



Figure 1: The Disease taxonomy

Table 1: The sample dataset

Case	Age	DBP	Disease	Category
1	50	90	Disease 1	A
2	56	90	Disease 2	A
3	52	101	Disease 3	A
4	50	95	Disease 3	B
5	56	97	Disease 3	B
6	(no data)	89	Disease 5	A
7	52	97	Disease 3	A

Phase 1b: Data import

As an alternative to data entry, Refiner++ can import data from other applications (databases and spreadsheets, such as Microsoft Access and Excel).

Since data type information is not saved in the CSV format, Refiner++ must infer the field data types. It does this by attempting to convert each value in a field into each data type; when a text value is parsed into a data type without errors, this is counted as a success for this data type. The data type with the most successes is selected as the type for that field. This process has proven to be very reliable.

In general, an expert would choose to enter data manually if he or she was using Refiner++ in an exploratory mode, and import data from an external source otherwise.

Phase 2: Generation of category descriptions

Refiner++ builds up a description of each category using the case values. The procedure used to create the category descriptions is as follows:

1. Take each category in turn, and select all the cases assigned to this category.
2. Take each field in turn, and make a list of all the values for this field for each case.

3. Use the list of values to create a generalized value (such as a range) which includes all these values. This is done in different ways, depending on the type of data in the field (see Table 2). The resulting generalized value is called a feature.
4. The category description is a collection of the features for each field.

Table 2: Generalization of Refiner++ data types

Data Type	Generalisation	Example
Numeric	Creates a range from the maximum and minimum values encountered	89 - 101
Date / time	Creates a range from the earliest and latest values encountered	17 th March – 5 th November
Taxon	Uses the nearest common ancestor	Dependent upon the taxonomy
String	Creates a list of all the encountered values	["One", "Two", "Three"]
Boolean	If all True, returns True If all False, returns False If we have a True and a False, the range returned is 'Any', signifying either True or False.	True, False or Any

As an example, consider the sample dataset given in Table 1. Category A has five cases assigned to it: 1, 2, 3, 6 and 7. To generate the Age feature we take the values for the Age field from each of these cases: 50, 56, 52, and 52. These values give us a range of 50-56 (the missing value in case 6 is ignored). Similarly, the values for the DBP field (90, 90, 101, 89, and 97) give us a range of 89-101. The Disease field is different: since it is a taxonomic field rather than a numeric field, the feature is generalized by finding the nearest common ancestor for each taxon. Consulting the taxonomy as given in Figure 1, the values obtained from these cases (Disease 1, Disease 2, Disease 3 and Disease 5) have the All Diseases node as their nearest common ancestor. Therefore, the description of Category A can be expressed as:

Age (50-56), DBP (89-101), Disease (All Diseases)

Using a similar procedure, we can express the description of Category B as:

Age (50-56), DBP (95-97), Disease (Disease 3)

Phase 3: Inconsistency detection

Once all the category descriptions have been created, Refiner++ checks for inconsistencies in the dataset. Each case is tested to check which category descriptions it matches, and an inconsistency is flagged for any case which matches a category it was not assigned to by the domain expert, or which does not match the correct category as assigned by the domain expert.

A case is deemed to match a category description when each of the fields for which it has data fit into the category description's value for that field. Missing values are considered to automatically match any value.

Returning to the sample dataset and the category descriptions we generated from it, we can see that cases 4 and 5 match category A, although they have been assigned to category B by the domain expert. Also, case 7 matches category B when it has been assigned to category A. Therefore, there are three inconsistencies in this dataset.

Phase 4: Inconsistency removal

If inconsistencies have been detected in the dataset, the algorithm attempts to suggest appropriate ways of dealing with the inconsistencies by refining the dataset.

Inconsistencies are caused by overlapping category descriptions, so we try to remove or minimise any overlap; two categories are distinct if even one of its features is distinct.

The various refinement strategies that Refiner++ suggests are:

- **Exclude a value from a feature:** if one category subsumes another in a particular feature, we might be able to create a disjunction to disambiguate them. For example, consider the category descriptions generated from the sample dataset; the description of category A allows it to match any cases with a value for DBP in the range 89-101, and B matches cases with DBP in the range 95-97 (in other words, B is subsumed by A for the DBP feature). If there were no cases in category A which fall into category B's range, we could create a disjunction by removing the subrange '95-97' from category A, leaving it as 'DBP = (89-94 or 98-101)'.
- **Change a value:** if one category subsumes another in a particular feature, but we cannot create a disjunction (see above) because the subsuming category contains cases which also match the subsumed category, we can try to move these cases out of the subsumed category's range. Continuing the example above, we can see that case 7 in category A has a value for DBP in the range 95-97, so we cannot create a disjunction; instead, Refiner++ asks the domain expert whether it is possible to change the case's DBP value so that it lies in the range 89-94 or 98-101 (Refiner++ looks at the other cases in category A to suggest likely alternate values).

Refiner is especially good at picking up typographical errors in datasets with this procedure, and uses this strategy to suggest alternative spellings and values for mistyped fields. In most other situations, however, this strategy is not appropriate.

- **Reclassify a case:** if a case matches a category other than the one(s) to which the domain expert has assigned it, this might indicate that the case has been misclassified. Refiner++ will suggest changing the case's classification to match that given by the algorithm.
- **Shelve a case:** if a case is classified incorrectly, Refiner++ will suggest that the case could be shelved. If the domain expert chooses this strategy, the case will no longer be considered during the validation phase (as if it belonged to none of the categories). This can be useful for 'problem cases' which might otherwise cause the algorithm to become stuck in a dead end.
- **Add a new field:** often, the data provided to Refiner++ is insufficient to produce unambiguous category descriptions, and so not every case can be classified correctly. When this occurs, Refiner++ suggests that the domain expert could add a new field to the dataset to allow such cases to be correctly classified.

Refiner++ typically suggests many refinement strategies each time it is asked to validate the dataset, and the domain expert is asked to choose the strategy he or she deems most suitable. In order to aid the domain expert's choice, the suggested strategies are ordered using two heuristics:

- Strategies which are suggested more than once appear at the top of the list, since they are likely to lead to the removal of the largest number of inconsistencies.
- Since we want to alter the dataset as little as possible, strategies are ordered by the number of cases they affect; strategies which affect fewer cases appear above strategies which affect many cases.

When the domain expert chooses a strategy to perform, the changes are made and Refiner++ re-validates the dataset, producing a new list of refinement strategies. This process continues until no inconsistencies are found.

Going back to the sample dataset and the inconsistencies we found in the previous step, Refiner++ suggests the following refinement strategies:

- Change the value of DBP in case 7 to 90
- Change the value of DBP in case 5 to 95
- Reclassify case 7 to category B
- Shelve case 7
- Change the value of Disease in cases 3 and 7 to "Disease 1"
- Reclassify cases 4 and 5 to category A
- Shelve cases 4 and 5

- Add a new field

In this case, suppose the domain expert chooses the option to reclassify case 7 to category B. Refiner++ will perform the change and validate the dataset again, producing the following suggestions:

- Remove the range 95-97 from the DBP feature of category A
- Change the value of DBP in case 4 to 97
- Change the value of Disease in case 3 to "Disease 1"
- Reclassify cases 4, 5 and 7 to category A
- Shelve cases 4, 5 and 7
- Add a new field

This time, the expert chooses to create a disjunction and remove the range 95-97 from the DBP feature of category A, which becomes (89-94 or 98-101). After performing the strategy and re-validating the data, Refiner++ reports that there are no more inconsistencies. The dataset has been successfully refined.

Changes to the Algorithm

There have been a number of changes made in this Java implementation of the algorithm, when compared to the earlier Refiner+ algorithm. These changes have been made for a number of reasons, primarily:

- to allow for the transition from LISP (the original implementation language) into Java
- to increase the algorithm's speed and efficiency
- to introduce new features which will be of use to end users

Secondary classification of cases

The original algorithm required the domain expert to specify which category each case belonged to, and also whether each case was a prototypical or non-prototypical example of the category. Since the secondary classification of cases as prototypical or non-prototypical required extra effort from the domain expert and was not a vital part of the algorithm, this distinction was dropped.

Removal of identical features

Once category descriptions are inferred, Refiner+ removed any features which were identical for all categories (such as the Age feature in the example above) since identical features cannot be used to discriminate between categories. In Refiner++, however, we recognise that useful strategies might involve these features, and so they are not removed.

Classification to multiple categories

One significant improvement Refiner++ makes over the original algorithm is that cases can now be assigned to multiple categories. Previously, a case could be assigned to only one category at a time. Having the ability to assign a

case to more than one category allows us to tackle a much greater variety of problems, and gives us new ways in which to deal with datasets.

New data types

Refiner+ was capable of using fields with numeric values and taxon values only; Refiner++ offers three additional data types:

- Boolean
- String
- Date / time

These new data types allow us to model a much wider range of datasets. Refiner++'s object-oriented architecture allows us to design and implement new data types as and when they are required. For details of new data types planned, see the section on 'Ongoing Work'.

Persistence

In Refiner++, datasets are persisted using industry-standard XML. The category descriptions are saved with the dataset, unlike in Refiner+, so the domain expert does not have to regenerate them.

Undo and Redo

The ability to undo and redo changes to the dataset allows users both to correct their mistakes and to perform a range of exploratory "what if...?" operations. This feature allows the domain expert to try out different approaches and back-track to the dataset's original state. In the current implementation, there is no limit to the number of strategies that can be undone.

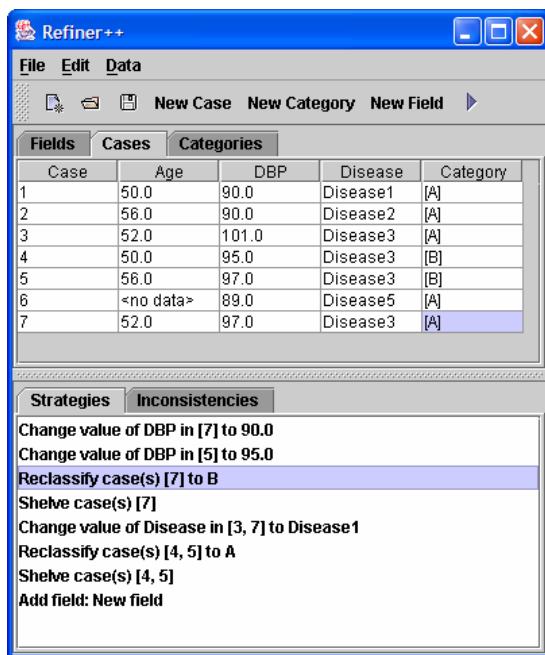


Figure 2: The Refiner++ main screen

The User Interface

As it is intended that the end users for Refiner++ should be domain experts and not knowledge engineers, it has been necessary to build the interface with particular care to make its functionality as transparent as possible.

With the new Refiner++ graphical user interface (see Figure 2), the domain expert can immediately see the fields, cases and categories which make up the dataset. From the Fields tab the user can easily add, edit and remove fields, along with any background information used by the fields (such as a taxonomy). From the Cases tab the user can see each case, their data values, and which categories they belong to. Cases can be added and removed here, their values can be edited, and they can be assigned to categories. On the Categories tab, the user is presented with a list of the categories in the dataset, along with their descriptions. From here the user can add and remove categories.

The lower section of the dialog contains the Strategies and Inconsistencies tabs. When a validation is performed, the user can see the list of refinement strategies produced by the algorithm on the Strategies tab, and the inconsistencies found by the algorithm on the Inconsistencies tab.

When a refinement strategy is double-clicked, Refiner++ automatically performs it and asks the user if they would like to re-validate the dataset.

USING REFINER++

We have had sessions with 3 domain experts where we used Refiner++ as the focus of the interaction; as we shall indicate the reactions to the system have been mixed. We report below a session with a physician who wanted to profile High Dependency Unit (HDU) patients as to whether they should be moved to another ward. Secondly, we met an Educational Psychologist who described a number of typical cases which she deals with, and finally with an anaesthetist who is a specialist in pain control. The first two experts were in the process of (formally) characterizing their domain of expertise; the third had already collected a set of cases as a database and wanted Refiner++ to review these for consistency.

Expert 1: Decisions as to whether a patient's level of care should be changed.

A consultant physician at the High Dependency Unit (HDU) at Aberdeen Royal Infirmary is in the process of defining protocols to determine whether patients should be moved to a different ward (either the General Ward or to the Intensive Care Unit (ICU)), or whether they should stay in the HDU for a further period. For the present, the clinician decided to look at 3 distinct groups of patients – those whose clinical problems were primarily Cardiac, Respiratory or Neurological.

Thus, if one categorizes the move from HDU to ICU as a move to a HIGHER level of care, a move from HDU to the general ward as a move to a LOWER level, and staying in

the HDU as the SAME level, then there are potentially nine categories of patients:

(HIGHER v SAME v LOWER) versus

(CARDIAC v RESPIRATORY v NEUROLOGICAL)

However, the decision was taken to model the Cardiac, Respiratory and Neurological patients as three separate datasets as it was felt that this would provide a more meaningful analysis. Having outlined his overall objective for the exercise, the consultant then turned his attention, as requested, to discussing individual cases. He started by describing normal Cardiac, Respiratory and Neurological patients – and here he noted the typical descriptors that he would record for such patients:

- Heart Rate (HR)
- Respiratory Rate (RR)
- Systolic Blood Pressure (SBP)
- Temperature (TMP)
- Glasgow Coma Scale (GCS), and the simpler AVPU scale²
- Urine Volume (UV)
- Oxygen Saturation (OS)

The consultant then specified 7 cardiac cases: 3 “higher” cases (those who should be moved to the ICU), 2 “same” cases (who should stay in the HDU), and 2 “lower” cases (who could be moved to a general ward). Refiner++ inferred descriptions for each of the three categories (Higher, Same and Lower), and showed that there were no inconsistencies in this dataset.

He then articulated 7 neurological cases where there was the same breakdown (3 higher level, 2 same level and 2 lower level cases) as before. These cases were also deemed consistent by Refiner++.

Table 4: Dataset for respiratory patients

Case	HR	RR	AVPU	OS	Cat.
1	105	27	A	94	Higher
2	120	35	V	88	Higher
3	140	45	P	80	Higher
4	105	28	A	94	Same
5	90	22	A	95	Same
6	80	18	A	96	Lower
7	70	15	A	98	Lower

² The AVPU scale describes a patient’s level of consciousness using the following four ranks: Alert, responds to Verbal stimulus, responds to Pain stimulus, and Unconscious.

Finally, 7 respiratory cases were articulated (shown in Table 4, with unused fields removed). Refiner++ generated the descriptions for the 3 categories as shown in Table 5. Refiner++ also reports two inconsistencies:

- Case 4 should not match category “Higher (to ICU)”, but does.
- Case 1 should not match category “Same (HDU)”, but does.

The system suggests 15 strategies for removing the inconsistencies, including:

- Reclassify case 4 to “Higher (to ICU)”
- Reclassify case 1 to “Same (to HDU)”

Table 5: Initial category descriptions for respiratory patients

Cat.	HR	RR	AVPU	OS
Higher	105-140	27-45	A-P	80-94
Same	90-105	22-28	A	94 - 95
Lower	70-80	15-18	A	96 - 98

The expert explored the several strategies suggested (using the UNDO facility), and finally decided to opt for the 2nd option above - namely to “Reclassify case 1 to Same (to HDU)”. This strategy removed the inconsistencies and produced the category descriptions shown in Table 6.

Table 6: Final category descriptions for respiratory patients

Cat.	HR	RR	AVPU	OS
Higher	120-140	35-45	V-P	80-88
Same	90-105	22-28	A	94 - 95
Lower	70-80	15-18	A	96 - 98

Interaction between the expert, Refiner++, and knowledge engineers

The expert was very forthcoming with actual stereotypical cases to represent the three medical conditions considered. From these he and the knowledge engineers decided the descriptions required, and one of the knowledge engineers defined these as fields within Refiner++. At this point the domain expert ‘dictated’ examples of each of the medical conditions for the three categories. Although the domain expert did not actually use the Refiner++ system, he was very active in suggesting how the information could be formalized.

NB: As this project is being actively pursued, we will soon have access to actual decisions made by clinicians about HDU patients’ movements. We plan to use this real-world

dataset to test and refine the category descriptions inferred here.

Expert 2: Classify severe educational psychology difficulties

We interviewed an expert in the domain of educational psychology. The focus of the interview was the process of diagnosing autism in pre-school aged children. The expert indicated that children with this condition / syndrome have (are defined to have) 3 features, known as triads:

- Problems with language and verbal communication
- Problems with social interaction (for example, play was either solitary or in parallel to that of other children, the normal being interactive play)
- Obsessive behavior (some examples being a marked aversion to wearing clothes with buttons, or performing repetitive hand movements)

We discussed three specific cases:

- A classically autistic child (case 1)
- A mildly autistic child (case 2)
- A child, probably not autistic, but with difficulties in language development (case 3)

The details of these cases are given below:

Case 1: Classical autistic patient

Andrew has limited language for his age, definite difficulties with syntax, and with the conventions of human-human communication. Further, he avoids eye-contact most of the time, and only plays on his own (solitary play). Additionally, he will only play with his favorite lorry, and eats only a limited range of foods which have to be arranged on his plate in a fixed sequence.

Case 2: Mildly effected autistic patient

Michael has limited language capabilities for his age and tends to use remembered phases which makes him appear “adult”. Further, Michael has unnatural eye-contact; however, he does play with other children but always copies what they are doing, he does not initiate new games / variants. Additionally, when stressed he grasps his favorite teddy bear (previously he had, under these conditions, also repeatedly shouted “Dinner now!”).

Case 3: Difficulties with language development

Richard has considerable language difficulties for his age – both in terms of the syntax of English and his range of vocabulary. However, although a slightly shy boy he does make fairly normal eye contact with people he interacts with. Play with other children is restrained but he does interact reasonably normally and does in a limited way innovate. He does have tantrums occasionally but does not display any repeated or obsessive behavior.

The domain expert, in discussions with the knowledge engineers, discussed at some length the general characteristics of the three categories. These have been documented but are not included here due to space limitations.

Interaction between the expert, Refiner++, and knowledge engineers

This expert seems to be uncomfortable with the idea of using a computer to capture her knowledge, but was very willing to provide us with examples of cases she had encountered and to discuss them more abstractly. At several stages the knowledge engineers articulated the descriptors associated with one of the triads, and then suggested that she describe one of her cases using these descriptors, but this was not pursued.

Expert 3: Pain tolerance (anaesthesiology)

Refiner++ was presented to an anaesthetist who had a pre-existing database which he was keen to use with Refiner++ to check the dataset for consistency.

This database contained data about patients who had been given epidurals; more specifically, it concerned the types of epidurals that were given and how effective the injection was (fields in the dataset included, for example, the level at which the injection was administered, the opiate used, whether the patient was receiving other analgesics, and whether an adjuvant was added to the epidural).

It was decided that we would use Refiner++ to categorize patients by the number of days the epidural was left in for. If an epidural does not seem to be working it is removed; in general, epidurals lasting for longer than a week are unusual. The possible categories are therefore 0, 1, 2, 3, 4, 5, 6 and 7 (0 if the epidural was removed the same day; 7 if it was removed after a week, indicating a very successful procedure).

The data was exported from Access into Refiner++ with no difficulty, but required some massaging (due to the presence of large freeform text entries in the database). This could be done with no difficulty using the Refiner++ UI.

When Refiner++ attempted to validate the dataset, the results were inconclusive; no useful strategies were suggested (the majority of the suggested strategies were of the Change Value type, which is not appropriate for imported medical data). However, the session with the expert was useful as some of the algorithm’s limitations were highlighted, and several improvements were suggested. There seem to be two main reasons for these difficulties:

- The categories were contiguous; this is not ideal, as cases could be on the border between two categories (see the section on Limitations below).
- The database was only used to store data about the epidural, and not about the patient. As such, it seems unlikely that any conclusions about the patient could be derived from this data source.

Interaction between the expert, Refiner++, and knowledge engineers

After a brief introduction to Refiner++ being run with the hypertension data, the domain expert seemed to have a good appreciation of its functionality. However, as he was deeply involved in porting the existing database from his PC to the one on which Refiner++ was running, he did not actually use the system.

CONCLUSIONS

The current version of Refiner++ is stable and has provided a very acceptable response³ with the datasets tried so far. We have carried out some scalability testing, the results of which are summarized in Figure 3.

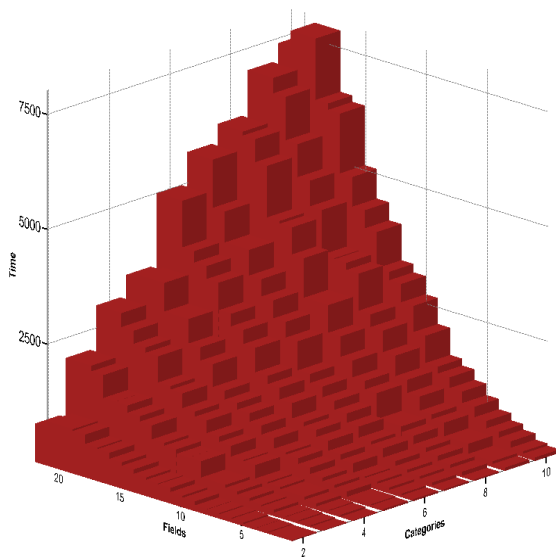


Figure 3: Scalability (fields and categories vs. time)

We have discovered that, although the application can be used to import existing datasets and perform analysis on them, its real strength seems to be for an expert who wants to conceptualize a domain where the inherent task is **classification**. Refiner++ requires the expert to articulate cases, specifying the descriptors they believe to be important in their domain. This causes the expert to conceptualise their domain, bringing out the hidden relationships between descriptors that might otherwise be ignored.

Although all the experts we interviewed were happy to oversee the usage of the application, none showed any inclination to use it themselves. This may have been because:

- They had not yet formulated the concepts for the domain.

³ The response time for typical datasets is too small to be measured.

- Refiner++'s UI was a little confusing for a new user; steps are being taken to considerably improve this aspect of the system.
- They were ill-at-ease with how the program worked. If, for example, a demo was available to show how the program can be used to create a sample dataset, this might be made easier. Several such demos are planned with the enhanced system.

We expect these changes to have a major impact on the domain experts' acceptance of the system.

Limitations

Although Refiner++ is a significant improvement on Refiner+, we have identified a number of the algorithm's limitations:

- When Refiner++ is used with a dataset that has been imported from an existing database, the 'Add a new field' refinement strategy is rarely useful. This is because when the expert exports the data into CSV format, he or she typically exports all the relevant fields from the original database, and therefore the user would have to add the values for this new field to each case manually. Obtaining this new data is not usually possible after the fact as the expert will usually not remember with any accuracy the values for each case (especially if the data was not thought important enough to store in the database in the first place). However, this strategy is useful when the expert is brainstorming with the program.
- In its current state, Refiner++ is an absolute system that does not cope well with uncertainty. Ranges of values are calculated naively with no consideration given to outlying values or clustering. An improvement to the current system might be to allow Refiner++ to classify cases using fuzzy logic – for example, "Case 4 is in Category A (70%) or Category B (30%)".
- The descriptors of a category can only be a series of conjoined features. Disjunctions cannot occur at this top level, but only within a feature, as in "DBP = (89-94) or (98-101)". This is clearly a lack and may cause domain experts problems when trying to model particular domains; we will monitor the impact of this feature.
- Categories must be discrete; since Refiner++ attempts to categorise cases absolutely there must be no overlap between categories. For best results the categories should be completely independent rather than part of a contiguous range.

ONGOING WORK

Besides the changes noted above we are planning to make the following enhancements to Refiner++:

- More data types are planned. In particular, ordered sets (e.g., low, medium, high) and enumerated sets (e.g., north, south, east, west).
- We plan to add a feature to allow the user to specify which strategies Refiner++ can suggest.
- The ability to specify that overlapping categories have a hierarchical relationship.
- We hope to produce a “refinement workbench” to include Refiner++, ReTax [1] and ConRef [9].

We plan to perform more studies of domain experts using Refiner++. As a result of these preliminary investigations we have made a significant number of improvements to Refiner++, particularly to the UI. We have also more fully appreciated the need to show domain experts a demo of a comparable domain being formulated with Refiner++ before inviting them to use the system. From our current perspective, the critical questions to be answered are:

- Given a reasonable introduction to Refiner++, will there still be experts who will be unable to formulate categories without support from a knowledge engineer? Our plan for future studies is to ask each domain expert to come to a session with written descriptions of cases, and then to review these descriptions to extract the important descriptors.
- Can this approach only be used with classification tasks?
- Can this approach only be used when each of the categories are distinct? Contiguous categories appear to be a problem.

RELATED WORK

John McDermott and his group at CMU had two important insights in the 1970s and 80s; firstly that KA systems should be directly useable by domain experts, and secondly that these tools should be built to acquire the knowledge required by particular problem solvers/problem solving methods. The principal reason being that the information required from the domain expert would then be more focused and hence easier to acquire. Initially, they implemented two KA systems targeted at classification and diagnosis, namely MORE and MOLE [4]. Subsequently, they developed a KA system, SALT, to collect information to drive a propose-and-revise (synthesis) PSM engine [5]. In essence they used this tool to collect knowledge for the VT (vertical transport) domain. Note that the above systems were generic, that is they were able to handle any domain. Mark Musen’s insight, captured in his OPAL system, was that domain experts would find the interactions more natural if they were conducted in the vocabulary of their own specialization. Thus OPAL incorporated the specialized domain vocabulary and allows users to interact through the forms which oncology specialists regularly use. In fact, the first version of PROTEGE created the OPAL KA tool, hav-

ing been provided with a specialist vocabulary and domain-specific graphical forms [6].

The approach taken in the RKF project [3] is considerably different, and in a sense considerably more ambitious. Basically this project defines components (including associated actions) which can be combined by domain experts to capture “chunks of knowledge”. As these components have to be retrieved from a library they need to have names which are easily recognized. They have focused on actions like MOVE, PENETRATE, etc and have used experience from WordNet, Roget’s Thesaurus and Longman’s Dictionary of Contemporary English to select these terms. Relationships between concepts need to be specified. Composition must have predictable semantics which is accomplished by defining a restricted composition language of relations and properties. These relations and properties have their own axioms defining what inferences will be drawn from the composition of components [2].

The initial group of experts chosen were biologists; a typical action in this domain is to describe an mRNA to move to the outside of a cell (Axioms associated with MOVE would note this is a specialization of MOVE, which is an EXIT). Several experiments were run to see if after training the Biologists were able to use the system for KA; in an experiment: biologists trained for one week and had eight weeks to encode the knowledge from a chapter of text in Cell Biology. Subsequently the KBS created was required to answer Questions from the end of the chapter. A summary of the experiment is that Biologists could find “Building Blocks” in the library and were comfortable defining biological processes in terms of generic actions such as COLLIDE, SLIDE, ATTACH and RELEASE.

ACKNOWLEDGMENTS

We would like to acknowledge support for the project from the EPSRC for the AKT IRC (GR/NI 5764).

The involvement of our three domain experts, Dr Brian Cuthbertson, Ms Helen Carmichael and Dr Brian Stickle was crucial to this work. Also, Ms Susan Frame has provided additional programming effort for Refiner++.

REFERENCES

- [1] Alberdi, E., Sleeman, D, “ReTax: A Step in the Automation of Taxonomic Revision”, *Artificial Intelligence* 91, 257-279, (1997)
- [2] Barker, K., Porter, B., Clark, P., “A Library of Generic Concepts for Composing Knowledge Bases”, *KCAP-01 Proceedings*, 14-21 (2001).
- [3] Clark, P., Thompson, J. et al, “Knowledge Entry as the Graphical Assembly of Components” *KCAP-01 Proceedings*, 22- 29 (2001).
- [4] Eshelman, L., “MOLE: A Knowledge Acquisition Tool for Cover-and-Differentiate Systems”, *Automat-*

- ing Knowledge Acquisition for Expert Systems, Kluwer Academic, Boston, 37-80 (1988).
- [5] Marcus, S., "A Knowledge Acquisition Tool for Propose-and-Revise Systems", Automating Knowledge Acquisition for Expert Systems, Kluwer Academic, Boston, 81-123 (1988).
- [6] Musen, M., "An Editor for the Conceptual Models of Interactive Knowledge Acquisition Tools", The Foundation of Knowledge Acquisition, Academic Press, London, 135-160 (1990).
- [7] Sharma, S., Sleeman, D., "Refiner: A Case-Based Differential Diagnosis Aide for Knowledge Acquisition and Knowledge Refinement", EWSL 1988, 201-210.
- [8] Winter, M., Sleeman, D., "Refiner+: An Efficient System for Detecting and Removing Inconsistencies in Example Sets", Research & Development in Expert Systems XII, 115-132 (1995).
- [9] Winter, M., Sleeman, D., Parsons, T., "Inventory Management using Constraint Satisfaction and Knowledge Refinement Techniques", Knowledge-Based Systems 11, 293-300 (1998).