VALIDATING CONCEPTS FROM AUTOMATED ACQUISITION SYSTEMS

Albrecht HEEFFER

University of Ghent
Applied Epistemology Lab
Blandijnberg 2, B-9000 Ghent, Belgium

ABSTRACT

Relevant domain features are used for the representation of knowledge in production rules and for the description of examples for rule induction programs. A concept acquisition CCAS, induces domain concepts for the description of middle-game positions in chess, and uses quantitative measures of information content for validating the acquired concepts.

I. Introduction

The main problem with the use of production rules to represent knowledge is the determination of the relevant features of the domain. Production system knowledge consists or rules of the form:

IF features $f_1, f_2, .. f_i$
are present in a situation
THEN apply action A

The features f_i must be provided to the system for recogni tion of a situation as relevant for the firing of rules. However, an enumeration of all relevant features is a very difficult task. Therefore, it is desirable and sometimes necessary to use inductive methods for establishing relevant features of the domain.

Chess is an excellent domain for experiments in concept acquisition. Chess players use a large vocabulary of domain concepts which is accessible in chess books and by psychological experiments. A grandmaster has a large library of chunks in LTM that is used to encode a chess position and in this way, reduces the uncertainty. It has been estimated that some ten thousands of chunks are necessary to explain the performance of master players [6].

A convenient method for measuring the amount of information reduction by chunking is the use of the binary question experiments. Jongman [2] proposed a method in which a chess player is allowed to ask questions about a chess position he has to reconstruct, that can be answered with only 'yes[1] or 'no'. The number of questions necessary to reconstruct the position correctly gives a good estimate of the information content of chess positions. Strong players (above 2200 Elo rating) require between 50 and 80 bits to reconstruct a position. For weaker players this amount may grow to above 100, depending on the type of position.

The intention of the experiment reported in this paper is to design a method for automated acquisition of chess concepts, that can be considered as chunks. These chunks can then be used as attributes for production rules or for further rule induction. The validity of the chunks will be checked by the binary question experiments.

II. THE ACQUISITION SYSTEM

A. Definitions

Attributes are used to describe objects. For chess positions we use 32 nominal attributes that categorically describe certain aspects of a chess position. Attributes represent the mapping of pieces on square locations:

```
attributes a_i with i=1,2,..,32
    = {wK,wQ,wqR,wkR,...,bfp,bgp,bhp}
DOM(v_i) = {a1,a2,a3,...,h7,h8,-}
```

The characters represent the well-known abbreviations of piece names used in chess notation. Capitals denote pieces, 'w' stands for white, 'b' stands for black and '-' indicates that the square is not occupied.

An instance is a description of an object as a vector of <attribute,value> pairs. In the experiments to be described, instances are middle-game positions of master games. The space of all syntactically correct instances is called the event space.

Selectors can be expressed in the annotated predicate calculus [3] as a form $a_i \# R_i$ in which , the reference is a disjunction of values of the domain of attribu a_i , and # stands for the operators '=' or '<>'.

A pattern is a logical conjunction of selectors. A pattern *pat*$_i$ is present in an instance if the attribute values of the instance satisfy all the selectors of the pattern. For example:

```
[bK=g8] & [bkB=g7] & [bfp=7] & [bgp=6] & [bhp=7]
```

A satisfaction factor f_s expresses the degree to which a pattern is satisfied by c_i ,the observed set of instances. The value of the satisfaction factor of a pattern is the proportion between instances that completely satisfy that pattern and c_i.

$$f_s(pat_n) = \frac{sat(pat_n, c_i)}{c_i}$$

This concept is similar to that of projected sparseness as defined by Michalski and Stepp [3].

A relevance list is a list of functions that maps keys onto a set of attributes that are relevant to form a pattern. It consists of elements of the following form:

$$k_1 :- (a_1, a_2, \ldots, a_i)$$

Keys are salient attributes of an event description. A relevance list is supplied by a domain expert to constrain patterns to "meaningful" ones.

A nilpattern is an <attribute,value> pair with a satisfaction factor higher than some threshold value. Nilpatterns can be detected by calculating $f_8(a_i)$ for all i, which are in fact all patterns consisting of one attribute. Events determined by nilpatterns with a threshold value equal to that of a single instance form the projected event space.

B. The induction algorithms.

The intention of an induction algorithm is to generate patterns with a satisfaction factor above a certain threshold, using the attributes defined in the relevance list. This Involves a search through a tangled hierarchy of three levels depth: the keys in the relevance list, attributes attached to each key and the nilpatterns of each attribute. As the algorithm uses a backtracking search for the attributes, the order of attributes in the relevance list is important. This implies that the most important attributes should be listed first. If no patterns are generated with the first attributes, no patterns containing the next attributes in the list will be considered. For the experiments reported in this paper, a permutation algorithm was used in which the order of the attributes is not important. All possible combinations of attributes in the relevance list, that include the key attribute, will be tried. The advantage of the permutation algorithm is that no relevant patterns will be missed because of ordering errors.

C. Implementation

The program, named CCAS (Chess Concept Acquisition System), is written in C-Prolog, running on a VAX 11-750 under UNIX. It consists of four basic components:

1. Nilpattern generator

Nilpatterns are the basic building blocks of chess concepts. They have to be generated from the set of given instances before the induction process can begin. Generating all nilpatterns, required 104 CPU seconds. Nilpatterns are represented as a functor with two arguments. The first argument is the attribute name, and the second is an ordered list of values, the first value having the largest satisfaction factor. An example of the generated nilpatterns for the white king:

```
nilpattern(wk,[g1,h1,h2]).
```

2. The satisfaction factor calculator

Both the nilpattern generator and the induction program rely on satisfaction factors of patterns. Efficient calculation of satisfaction factors is therefore important for the overall performance of the program. Using Prolog's unification for matching, the calculation of satisfaction factors is considerably faster than using other matching strategies. Calculating the satisfaction factor of a pattern with 10 instantiated attributes takes about 150 msec for a database containing 54 instances.

3. The induction program

The generation of all patterns for a relevance list with six attributes attached to a key takes about 150 CPU seconds for each key. Execution times grow exponentially with the number of arguments. For this reason, it is recommended to use many keys with lesser attributes, rather than to look for patterns with many attributes. This strategy is well suited for the domain of chess, but could be a constraint for other domains.

4. Binary question generators

Chess concepts induced by CCAS are operationally defined as valid if they reduce the information content of a chess position. Although the validity of generated ooncepts can be appreciated by a domain expert, the program also facilitates techniques for quantitative evaluation of the output. It includes a binary question generator that is able to reconstruct a chess position by asking questions about piece-square relationships. It uses nilpatterns for question generation.

III. RESULTS

Using the following relevance list

```
rel(bK,[bkR,bkB,bep,bfp,bgp,bhp]).
rel(wK,[wkR,wkN,wep,wfp,wgp,whp]).
rel(bdp,[bap,bbp,bcp,bqB,bqR]).
rel(bdp,[bcp,bep,wcp,wdp,wep,wfp]).
rel(wbp,[wap,wcp,wqR,wqN,wqB,wkB]).
```

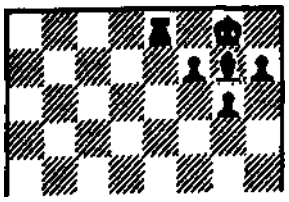
and 54 instances taken from 21st move positions in a chess book ([7] games 25 to 75), the system generated 275 patterns. The number of attributes per pattern varies from 1, the nilpattern of the key, to 7, the key and all attributes in the relevanoe list. The most Important patterns are the most specific ones, i.e. the patterns using the largest number of attributes. For each key in the relevance list, there is at least one such pattern. Most other patterns are subsets of these most specific ones.

The binary question experiment was tried for two types of positions, those included in the list of instances, and others that are similar, but not included. The program achieves a reconstruction performance of 100% for positions included in the instances, using an average of about 60 questions. For chess positions not included in the list of instances 100% reconstruction is not guaranteed but often reached, within the same average number of questions. These tests indicate that the system performs on the level of expert chess players.

IV. DISCUSSION

The concept acquisition system, described in this paper, is an attempt to fill a gap in the current spectrum of automated acquisition systems. The problem addressed in this project cannot be approached by any of the existing methods. In fact, generating concepts for the description of chess positions is something that is presupposed by existing learning programs such as ID3 [8] and CLEAR [4], Taxonomic classification programs such as CLUSTER [33 that determine hierarchies of subcategories within a collection of objects are more- closely related to the problem of concept acquisition but still not suited. All rule induction programs require a set of relevant attributes to describe the domain objects. But the construction of relevant attributes for representing instances is a difficult task itself.

The patterns generated by the program can very well be considered as concepts. They are human-intelligible and some even have terms attached to them that are used in the vocabulary of domain experts, such as kings fianchetto.



ACKNOWLEDGEMENTS

This work was carried out during a three-month visit at the Turing Institute [1]. Financial support was provided by the Belgian NFWO and the ICIWO. I would like to express my thanks to Prof. F. Vandamme of the University of Ghent, and to Prof. Donald Michie, Mr John Roycroft and Dr Tim Niblett of the Turing Institute, for their helpful suggestions.

REFERENCES

[1] Heeffer A. "Automated acquisition of concepts for the description of middle-game positions in chess", Turing Institute Research Memorandum TIRM005. December 1984, Glasgow.

[2] Jongnan H. Het oog van de meester, Van Gorcum, Asse, 1968.

[3] Michalski R. S, Stepp R. "Learning from observation: Conceptual Clustering", Chapter 11 in Michalski, Carbonell and Mitchell (eds) Machine Learning: An Artificial Intelligence Approach, Tioga Publishing Co., Palo Alto, 1983.

[4] Niblett T. "An interactive rule induction system" MINews 2, 1983, 8-15.

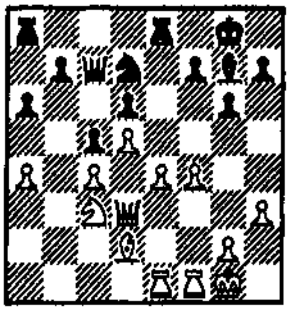
[5] Quinlan J. R. "Semi-autonomous acquisition of pattern-based knowledge", in Hayes, Michie and Pao (eds) Machine Intelligence 10, Ellis Horwood, Chichester, 159-172, 1982.

[6] Simon H., Gilmartin K. "A simulation of memory for chess positions", Cognitive Psychology 5, 1973t 29-46.

[7] Zeuthen S. Modern Benoni: survey of a structure.Zeuss Transactions, Denmark, 1973.

APPENDIX

A typescript of a binary question session shows how CCAS tries to reconstruct the position from the diagram. The program was not able to reconstruct the position 100% correctly, because it had no previous experience with a white queen on d3. Adding this position to the list of instances would solve the problem in future sessions. Apart from the problem with the queen, the reconstruction performance is quite well. The system reached a 95% correct reconstruction after only 46 questions.



```
1. wK-g1? y
2. wQ-f3? n
3. wqR-a1? n
4. wkR-f1? y
5. wqB-c1? n
6. wkB-d3? n
7. wqN-c3? y
8. wkN-(—)? y
9. wap-4? y
10. wbp-2? n
11. wcp-0? n
12. wdp-5? y
13. wep-4? y
14. wfp-4? y
15. wgp-2? y
16. whp-2? n
17. bK-g8? y
18. bQ-d8? n
19. bqR-a8? y
20. bkR-e8? y
21. bqB-c8? n
22. bkB-g7? y
23. bqN-(—)? y
24. bkN-(—)? n
25. bap-6? y
26. bbp-7? y
27. bcp-5? y
28. bdp-6? y
29. bep-0? y
30. bfp-7? y

31. bgp-6? y          44. wkB-(—)? y
32. bhp-7? y          45. wcp-4? y
33. bkN-f6? n         46. bkN-d7? y
34. bqB-(—)? y        47. wQ-(—)? n
35. bQ-c7? y          48. wQ-f2? n
36. whp-3? y          49. wQ-e4? n
37. wcp-3? n          50. wQ-b3? n
38. wbp-0? y          51. wQ-d2? n
39. wkB-c4? n         52. wQ-d4? n
40. wqB-d2? y         53. wQ-e2? n
41. wqR-e1? y         54. wQ-e1? n
42. wQ-d1? n          55. wQ-b6? n
43. wQ-c2? n
```