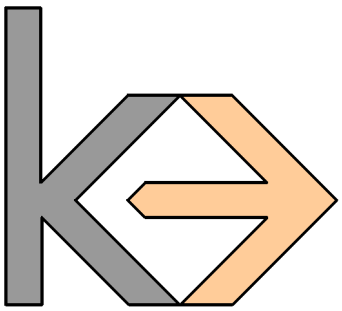


Diploma thesis

KRK Chess Endgame Database Knowledge Extraction and Compression

Gabriel Breda
Tutor: Prof. Dr. Johannes Fürnkranz

July 2006

Technische Universität Darmstadt
Fachbereich Informatik
Knowledge Engineering
Prof. Dr. Johannes Fürnkranz

Declaration of honour

I affirm hereby that the present diploma thesis was produced without help from a third person and only with the given sources. Every passages which were taken from sources are specified as such. This work was presented until now to no exam commission, neither in this form nor in a similar one.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, Juli 2006 Gabriel Breda

Contents

Abstract

The chess endgames are a perfect test domain for Machine Learning algorithms. Reciprocally, Machine Learning offers efficient methods to analyse and compress chess tablebases. Such databases consist of informations at a low level of description and are often huge, what makes them unpractical. Our goal is to reduce the size of the King and Rook against King (KRK) database without information loss with the help of knowledge-based compression. Some studies already tried in the case of the KRK endgame to use specific patterns in order to build a knowledge with a higher level of description. Grouping such patterns together, we transform the primary tablebase in a more compact form containing only values of these attributes. The representation of this new database as a decision tree simplifies its description, enabling a better compression rate than standard mechanical methods.

Chapter 1

Introduction

The game of chess is one of the oldest and most complex game in the history of humanity. With a simple board, thirty two pieces and a few rules it achieves a complexity level that the current computers cannot master.. The huge amount of possibilities in the game make it a perfect test domain for Artificial Intelligence, and especially for Machine learning: if we are able to invent a computer program that can learn to play such a difficult game, then we can also create similar ones that can solve a large part of real life problems which usually have a much smaller complexity. Reciprocally, Machine Learning algorithms can be testet very well on chess problems. Actually, the chess endgame domain is already complex enough to be a standard test place for Machine Learning algorithms. In particular, the compression of data using knowledge-based methods coming from the Artificial Intelligence domain can be perfectly studied on chess endgames tablebases, which provide large amounts of information at a very low level of description.

Our work is precisely engaged in that subject: Having a database which lists all existing positions of a 3 figures endgame, is it possible to compress it with knowledge extraction methods? Could another form of the database expressed by concept attributes from the chess domain be smaller than the primary one? This is the topic we study in the present report, in the case of the King and Rook against King endgame. After explaining in the first chapter some basis concepts of the domain and our precise procedure, we summarize the literature our work is based on. We then look in detail at the new knowledge which we build to transform the database and obtain new ones, whose properties we analyse afterwards in the fourth chapter, before transforming the databases again to obtain the optimally compressed form presented in the fifth chapter.

1.1 Chess Programming

1.1.1 Complexity in Chess Programming

The main problem of chess programming is the very large number of continuations involved. In an average position there are about 40 legal moves. If you consider every reply to each move you have $40*40 = 1600$ positions. This means that after two ply (half-moves), which is considered a single move in chess, 1600 different positions can arise. After two moves there are 2.5 million positions, after three moves 4.1 billion. The average game lasts 40 moves. The number of potential positions is to the order of 10^{128}, which is vastly larger that the number of atoms in the known universe (about 10^{80}). The exhaustive listing of all possible chess games is therefore impossible. Nevertheless this process is possible if only few pieces stay on the chess board at the end of a game, i.e. in chess endgames.

1.1.2 Common Endgames

The simplest endgames contain 3 pieces. A two pieces endgame has no interest, because the two pieces are the two kings and one king alone cannot take advantage over the other, so the game is drawn. The number of positions in an endgame database is to the order of 60^N, where N is is the number of pieces. The combinatorial explosion makes an enumeration impossible for endgames of more than a few pieces. The complexity of common endgames is listed in the table 1.1.

No. of pieces	Complexity
3	262 144
4	1 677 216
5	1 073 741 824
6	68 719 476 736

Table 1.1: Amount of positions in chess endgames

Endgame databases were originally constructed by Thompson ([Tho86]). They exist for all 3, 4 and 5 pieces endgames. Some have already been generated for 6-pieces endgames, but they are very complex. This complexity can be reduced with basic rules: Two pieces cannot stand on the same square; or basic chess knowledge: The two kings cannot be on adjacent squares (because a king cannot put itself in chess). With these rules, some theoretical possible positions are illegal

and so they don't have to appear in the database. Furthermore, depending on the nature of the pieces that exist in the endgame, some symmetries can also reduce the complexity. Particularly, endgames without pawns have no concept of direction (since the pawn is the only piece in chess that *goes forward*, all the other pieces can move the same in any direction). That's why pawnless endgames have a smaller complexity, as more symmetries can be applied to reduce the number of interesting possible positions.

Common 3-pieces endgames are King and Pawn vs King (KpK), King and Queen vs King (KQK), King and Rook vs King (KRK, which we will consider here) or King and Knight vs King (KNK). In the 4-pieces category there are for example King, Knight and Bishop vs King (KNBK), King and Queen vs King and Rook (KQKR) or King, Knight and Pawn vs King (KNpK). As a 5-pieces chess endgame we can cite King and two Knights vs King and Pawn (KNNKp). The basic endgames are finished by the end of the game: By check mate (if a side has its king in check and cannot do anything to avoid it), or draw (ie stalemate, if a side has no legal move to do but its king is not in check, or if checkmate is impossible, i.e. for example if only the two kings remain on the board). Some more complex endgames can be considered as ended by getting into a simpler endgame, like KQKR ends as KQK if the black rook is taken. Such endgames are then also called *converted*.

1.1.3 Endgame Databases

Different types of databases

Often also called tablebase, a chess endgame database is an ordered list of all positions in the endgame with interesting calculated values. There are different types of endgame databases. They usually list *black-to-move* positions, as the strongest side is often the white side, so the black side is mated in the simplest positions in the database, but they can also list *white-to-move* positions. All types of databases know whether a given position in the endgame is a win, loss or draw. If that is all the database contains, it is called a WLD-database (Win/Lose/Draw). If the database contains information on how long it will take until the game is over, it is called a distance-to-mate (DTM) database. This depth of win is calculated in case of an optimal play. If it contains only the information on how long it will take until a conversion takes place, it is called a distance-to-conversion (DTC) database. A conversion is either a pawn promotion or a piece being captured or checkmate. If a piece is being captured or a pawn promoted,

we come either to a draw if only the kings remain, or we have to look in another database with the new pieces set. These databases do not take into account the *fifty moves rule*[1]. To overcome that deficiency, Distance to zeroing (DTZ) and the superior Distance to rule (DTR) databases have been created, but only in theory.

The WLD database is smaller, but it has the problem that even though a program may be in a winning position, it might not be able to actually win the game. All the database tells is that it has a win, and it also tells which moves conserve the win. But some win-conserving moves may increase the distance to mate, and the program cannot easily decide which of these win-conserving moves to make. DTM databases are obviously better in this respect, since you just make the win-conserving move with the lowest DTM associated. DTC databases also solve the problem of winning a won position, however the program might take longer than necessary to do so. The main advantage of the WLD database is its size: storing WLD information only needs little space, therefore larger parts of the database can be kept in memory, if the database size is larger than the amount of memory of the computer (which is typically the case). Accessing the database when it is not loaded in the main memory is not really an option, as the speed is very slow compared to memory.

The best known databases are the ones of Ken Thompson ([Tho86]), Steven J. Edwards ([Edw95]) and Eugene Nalimov ([NWH99]). They are tablebases of type DTC, DTM and compressed DTM respectively. Most commonly used are those from Nalimov since they are free and more efficient. Nalimov tablebases are nearly “perfect" since they take into account *en Passant*[2]. However, they don't take *Castling*[3] into account, but it is usually ignored in a tablebase, because games in practice rarely reach the endgame without a king or rook moving, so castling almost never make sense here.

[1] The *fifty move rule* in chess states that a player can claim a draw if no capture has been made and no pawn has been moved in the last fifty consecutive moves.

[2] *En passant* is a maneuver in the board game of chess. The *en passant* rule applies when a player moves a pawn two squares forward from its starting position, and an opposing pawn could have captured it if it had only moved one square forward. The rule states that the opposing pawn may then capture the pawn as if it had only moved one square forward.

[3] *Castling* is a special move in the game of chess involving the king and one of his original rooks. It consists of moving the king two squares towards a rook, and moving the rook onto the square over which the king crossed.

Generation of chess databases

Endgame chess databases are generated via induction. The generation is easy if you have a good algorithm, but it can require huge amount of time and computer memory depending on the complexity of the databases which are generated. The generation consists in the following steps:

1. All possible positions are examined, and positions where one side is mated are marked. These are called *mated in 0* positions.

2. Once this is done, all positions are examined again, except those which are already marked, and if any of them can reach a "mated in 0" position, it is marked as *mate in 1*. The concept *possible position* here depends on the side which has the move: a position in which the black king is in check is not a legal white-to-move position.

3. Now we look at everything again, and the possible positions from which all possible moves lead to a "mate in 1" position are marked as *mate in 2*. It could seem weird to call these positions mate in 2, because mate appears after only 2 plies, but it is due to the fact that the loosing side has the move.

4. Now we look at all the possible positions again, and we try to reach "mate in 2": The found positions are also marked "mate in 2".

5. The next step is a bit different: We look for cases where it is impossible to avoid "mate in 2" and "mate in 1" positions. These are marked *mate in 3*.

6. And so on, until no progress is made. The rest of the positions are proven to be draws.

The last step can be difficult to evaluate, because the concept "no progress is made" depends on possible conversion. For instance in the endgames with pawns, if a pawn converts, you can end in several different other endgames, what brings up new possibilities. For example, there might be a mate in 105 moves in KBP vs KN, but there might not be a mate in 95. This is because the mate in 105 moves might involve immediate conversion to a KBN vs KN.

1.2 Database Size Reduction

The utilisation of chess endgame databases require them to be loaded in the main memory of a computer. Since most of these databases are too big for the memory, they have to be reduced. As the database has to be accessed as fast as possible, the general data compression methods or any method that slows down the access time cannot be applied. The aim is to find a more compact representation of the database without information loss.

1.2.1 Reduction Through Symmetries

The first and simplest reduction of the complexity is achieved through properties particular to the chess game. The board is the same in every direction, and most of the pieces move the same way forward and backward (in fact all pieces except pawns). That is why the interesting part of the database can be reduce thanks to various symmetries, which are more numerous if there is no pawn in the endgame. To do that, we build position equivalence classes. Each class is represented by a canonical position. All the positions that can be retrieved by rotating or reflecting a canonical position belong to the same class. Only the canonical positions are stored in the database. The exact symmetries and rotations which can be used to determine the canonical positions depend on the pieces of the endgame. We will explain more precisely for the KRK ending.

The KRK endgame is quite simple and its pieces offer good symmetric properties, since the only three figures in the endgame are the white king, the white rook and the black king, and they all move symmetricaly in all direction. These symmetries are resumed in the figure 1.1.

All the starting positions can be limited to those having the white king in a ten squares octant (a1, a2, a3, a4, b2, b3, b4, c3, c4 and d4). To obtain that, the following rules are applied (in that order):

1. **WK horizontal reflection**: All the positions in which the white king is on the right part of the board (file e or greater) are reflected through the central vertical axis so that it stands then on the left part of the board.

2. **WK vertical reflection**: All the positions in which the white king is on the upper part of the board (rank 5 or greater) are reflected through the central horizontal axis so that it stands then on the lower part of the board.

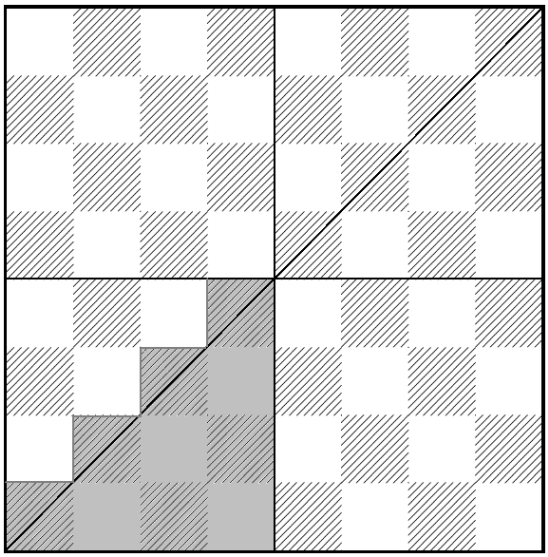


Figure 1.1: Symmetries used to reduce the KRK database

3. **WK diagonal reflection**: All the positions in which the white king is above the diagonal a1 to h8 are reflected through the diagonal axis so that it stands then below the diagonal.

4. **BK reflection**: All the positions in which the white king is on the diagonal a1-h8 and the black king is above this diagonal are reflected through the diagonal axis so that the black king stands then below the diagonal.

5. **WR reflection**: All the positions in which the two kings are on the diagonal a1-h8 and the rook is above this diagonal are reflected through the diagonal axis so that the rook stands then below the diagonal.

As summarised in the table 1.2, these symmetry operations reduced the database to 28056 positions. Compared to the rough KRK database, they make it almost 8 times smaller.

Rough 3 Figures database:	262 144 positions	(64*64*64)
Rough KRK database:	223 944 positions	(only legal positions)
KRK database:	28 056 positions	(with symmetry reductions)

Table 1.2: Amount of positions in the different forms of the KRK database

1.2.2 Knowledge-based Reduction

The research of higher level knowledge can help reducing the size of the database. For example, a rule that summarises many lines of the database takes less memory to be stored than all the corresponding lines, even if some exceptions also have to be stored. Depending on the size of a line and the number of exceptions, the induced rule is more or less efficient to compress the database. In our case of the KRK endgame for example, every position where the white rook can be taken is a draw. So, a rule characterising this fact can replace a lot of positions, and help reducing the database.

Some systems that perform Data Mining Compression exist already. For example, SPARTAN ([BGR01]) is such a system that exploits attribute semantics and data-mining models to effectively compress massive data tables. It takes advantage of predictive correlations between the table attributes and the specified error tolerances to construct concise and accurate *Classification and Regression Tree* (CaRT) models for entire columns of the table. The key is to achieve a good compression while staying at a tolerable computation time. To restrict the huge search space of possible CaRTs, SPARTAN explicitely identifies strong dependencies in the data by constructing a Bayesian network model on the given attributes, which is then used to guide the selection of promising CaRT models. The CaRT-building component also employs integrated pruning strategies that take advantage of the prescribed error tolerances to minimise the computational effort involved.

Our method is similar to the one of SPARTAN, but best adapted for our case. We used attributes which were partly automatically detected by existing systems (described in chapter 2). They are generally invented by systems which are helped by chess masters: the system is fed with chess positions, and depending on what it already learnt, it either saves a new pattern or learns a new rule. All these operation are based on Logic Programming (ILP) methods and Data Mining algorithms.

1.3 Data Mining

Data Mining, also known as Knowledge-Discovery in Databases (KDD), is the process of automatically searching large volumes of data for patterns. It is a fairly recent and contemporary topic in computing. However, Data Mining applies many older computational techniques from statistics, machine learning and pattern recognition. It can be defined as "The nontrivial extraction of implicit,

previously unknown, and potentially useful information from data"[FPSM92] and "The science of extracting useful information from large data sets or databases" [HMS01]. Although it is usually used in relation to analysis of data, data mining is an umbrella term and is used with varied meaning in a wide range of contexts. It is usually associated with a business or other organisation's need to identify trends.

Once we have built our new database with patterns which may help significantly in the prediction of the distance of win, we need tools to analyse the database and test which dependencies exist between the pattern values and the distance of win. The Data Mining should provide us with some tools to achieve this goal. We especially use Decision Trees to predict the depth of win of a position as efficiently as possible.

In machine learning, a decision tree is a predictive model, that is, a mapping of observations about an item to conclusions about the item's target value. The machine learning technique for inducing a decision tree from data is called decision tree learning. It is commonly based on the concept of Top Down Induction of Decision Tree (TDIDT, see [Qui86]). Here, a decision tree describes a tree structure wherein leaves represent classifications and branches represent conjunctions of features that lead to these classifications. A decision tree can be learned by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner. The recursion is completed when splitting is either non-feasible, or a singular classification can be applied to each element of the derived subset.

Amongst other data mining methods, decision trees is a method that has several advantages: Decision trees are simple to understand and interpret. People are able to understand decision tree models after a brief explanation. They require little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. They can handle both nominal and categorical data. Other techniques are usually specialised in analysing datasets that have only one type of variable. For example, relation rules can be used only with nominal variables while neural networks can be used only with numerical variables. They make it possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model. They are robust, perform well with large data in a short time. Large amounts of data can be analysed using personal computers in a time short enough to enable stakeholders to take decisions based on its analysis.

The open source Data Mining software WEKA[4][WF05] allows to apply lots of algorithms on data, like decision trees. That is the software we used in our work to analyse our database. In particular, we used the c4.5 algorithm.

c4.5 is a software extension of the basic decision tree generating algorithm ID3. ID3 (short for Iterative Dichotomiser 3) is a basic algorithm from Ross Quinlan to build decision trees. The ID3 algorithm can be summarised in three steps: It first takes all unused attributes and counts their entropy concerning test samples. Then it chooses one of the attribute with the smallest entropy, and eventually makes a node containing that attribute. The c4.5 algorithm contains several improvements, especially needed for software implementation. The improvements contain the possibility to choose an appropriate attribute selection measure, a way of handling training data with missing attribute values. They also allow the handling of attributes with differing costs and continuous attributes. A decision tree can also be pruned or not. Pruning a decision tree means simplifying it by removing some parts. Another important property of a decision tree is the minimum number of instance per leaf (short `minNumObj`). This parameter defines the level of detail of the tree. See [Qui93] and [Mit97] for more details.

1.4 Our Approach

The aim of this work is the compression of the KRK endgame database with the help of attributes. We worked on the standard KRK database with 28056 positions distributed as shown in table 1.3. The attributes should represent a knowledge of higher level than pure position description. They can be combined in rules, that are automatically build by a Data Mining Program named WEKA to optimise efficiency.

The first step was to find attributes and to implement them in a script. The script was implemented in ruby[5] and designed to execute the transformations of the attributes practically and efficiently. The source code is available in Appendix B. The attributes are grouped together to form theoretically a consistent set, so that the attributes in the set represent the initial KRK database as good as possible. In fact we have used several sets and then compared the results. Each set of attributes is of the form:

$$\mathsf{S} = (\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n)$$

[4] http://www.cs.waikato.ac.nz/ml/weka/
[5] http://www.ruby-lang.org/

Depth	Positions	Depth	Positions
draw	2796	8	1433
0	27	9	1712
1	78	10	1985
2	246	11	2854
3	81	12	3597
4	198	13	4194
5	471	14	4553
6	592	15	2166
7	683	16	390

Table 1.3: Distribution of the depths of win in the KRK database

Each attribute $\mathcal{A}$ is in fact a transformation that returns a value v from a position vector P:

$$P = (wkf, wkr, wrf, wrr, bkf, bkr, dow)^{6}$$

$$v = \mathcal{A}(P)$$

The distance to win is in fact not used to calculate the attributes' values, but it is written in the original database besides the positions.

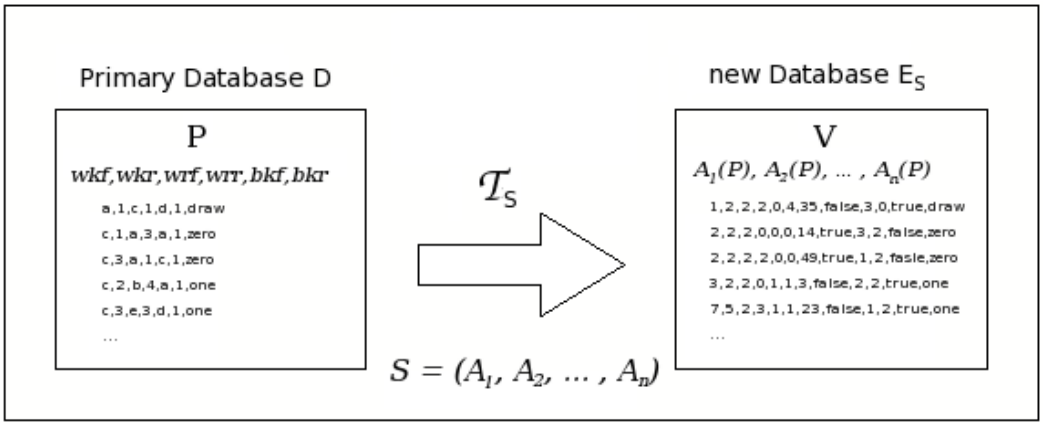


Figure 1.2: General approach: step 2

The second step was to build a new database for each set of attributes, and to check its properties. The new databases are composed of combinations of

[6] white king file, white king rank, white rook file, ..., depth of win

attributes' values, with a distance to win. Let $\mathtt{D}$ be the original database with the positions P of the pieces on the board: $P \in \mathtt{D}$ For a set of attributes S, the vector P is transformed into a vector V so that:

$$\forall i \in [1, n], v_i = \mathcal{A}_i(P)$$

$$V = (v_1, v_2, \ldots, v_n) = (\mathcal{A}_1(P), \mathcal{A}_2(P), \ldots, \mathcal{A}_n(P)) = \mathsf{S}(P)$$

As the depth of win should appear at last in the new database, we have

$$\mathcal{A}_n\Big((wkf, wkr, wrf, wrr, bkf, bkr, dow)\Big) = dow$$

The new database $\mathtt{E}_\mathsf{S}$ is then the list of all the combinations V:

$$\mathtt{E}_\mathsf{S} = \{\mathsf{S}(P), P \in \mathtt{D}\}$$

So we define a transformation $\mathcal{T}_\mathsf{S}$ from $\mathtt{D}$ to $\mathtt{E}_\mathsf{S}$, like drawn in figure 1.2.

$$\mathtt{D} \xrightarrow{\mathcal{T}_\mathsf{S}} \mathtt{E}_\mathsf{S}$$

Depending on the attributes' set, it can be that 2 different positions P_1 and P_2 with a depth of win dow_1 and dow_2 have the same image through all the attributes of the set except the last one (that encodes the depth of win):

$$\forall i \in [1, n-1], \mathcal{A}_i(P_1) = \mathcal{A}_i(P_2)$$

If the depths of win are the same ($\mathcal{A}_n(P_1) = \mathcal{A}_n(P_2)$), which should be the case if the attributes perfectly describe the board, the 2 positions are summarised by one image:

$$dow_1 = dow_2 \implies \mathsf{S}(P_1) = \mathsf{S}(P_2)$$

But if they are different ($\mathcal{A}_n(P_1) \neq \mathcal{A}_n(P_2)$), we have the following problem: 2 different positions with distinct depths of win are described by the same combination of attributes' values. This may be solved with the help of exception handling. Since we need to be able to retrieve perfectly the initial database, we have to store separately the exceptions, so that the transformation is a bijection. To achieve that, we extract from the initial database all the positions which would lead to an exception and store them in a separated file $\mathtt{D}_{err}$. The original database without these positions is called $\tilde{\mathtt{D}}$, and the transformation $\mathcal{T}_{\mathsf{S}|_{\tilde{\mathtt{D}}}}$ from $\tilde{\mathtt{D}}$ to $\tilde{\mathtt{E}}_\mathsf{S} = \mathcal{T}_\mathsf{S}(\tilde{\mathtt{D}})$ is a bijection.

$$\tilde{\mathtt{D}} \xrightarrow{\mathcal{T}_\mathsf{S}} \tilde{\mathtt{E}}_\mathsf{S}$$

So the new database is then composed of two files: The error file containing all the ambiguous positions, and the real new database with the attribute values, which will then be compressed with data mining algorithms. Our aim is of course to have an error file as small as possible, since it cannot be compressed efficiently (only standard compression like zip). In the perfect case, all the positions have a unambiguous image, so there are no errors.

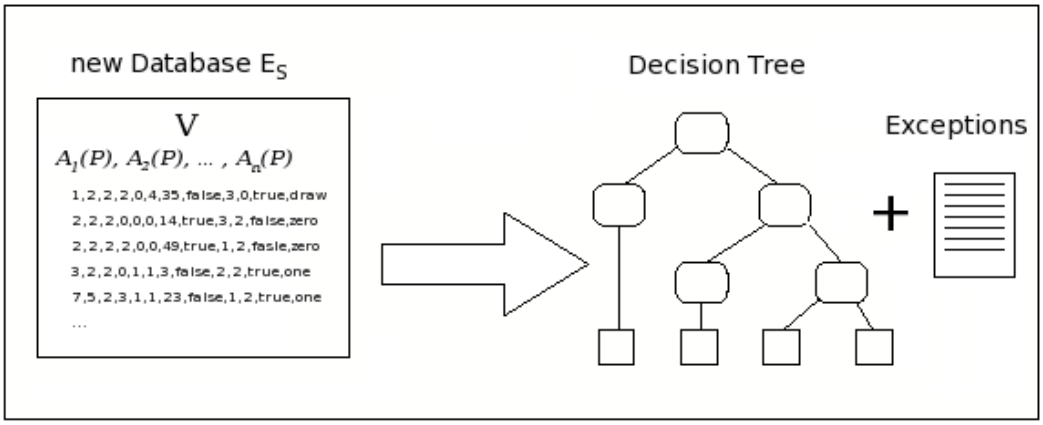


Figure 1.3: General approach: step 3

The third step consisted in trying data mining algorithms to compress the databases. It is illustrated on figure 1.3. We worked with decision trees, in which the depth of win of the pattern value combinations in the new database is analysed. As the tree can generalise and regroup some attribute combinations together, it needs less disc space when saved in a file than the database. Moreover the properties of the tree can be adjusted to have the best results. But it can happen that some instances are incorrectly classified in the tree, depending on its properties. Therefore we need to store beside the tree a list of the incorrectly classified combinations.

Chapter 2

Review of related work

The KRK endgame was already the subject of a lot of studies, especially in Machine Learning. Some of these studies particularly deal with patterns, which are supposed to represent some chess knowledge, what is exactly our subject. Our study is based on three papers and on the description of a chess playing machine. These four sources contain interesting elements for our work, therefore I will give an overview of their content. Please refer to the original papers for more details.

2.1 Generalising Closed-World Specialisation: A Chess End Game Application, by M. Bain ([BMS95])

In his paper Bain describes his experiments with a system based on an algorithm named Closed-World Specialisation (CWS). He tested his system on the KRK chess endgame. His aim was to develop an agent that, given access to the complete list of pre-classified positions, can learn rules to correctly classify each position by the optimal number of moves to checkmate, or as drawn. This system is based on a few simple patterns, and creates new ones to build recognisers for each depth of win.

2.1.1 Introduction

Theory discovery is a branch of symbolic learning that has a good application in chess endgames: An agent has to learn from a chess database a set of rules to

classify each position of the endgame. The rules are expected to be more compact than the database and have to be transparent to a human. The example of KRK typically offers a large amount of data at a low level of description, from which a simple set of comprehensible rules with good predictive accuracy has to be learned

In many real-world problems the size of the database grows continuously even after the start of the learning phase of the algorithms. That leads to possibly too big databases for batch learning. Therefore a sampling approach is needed as well as the possibility for the learned theory to change continuously. Thus Bain and Muggleton were interested in incremental learning systems, with methods of correcting theories in a framework of minimal specialisation. It was shown that minimal correcting specialisations are not always obtainable within classical logic. Consequently they presented an algorithm carrying out a specialisation scheme based on a non-monotonic logic formalism used in logic programming: Closed-World Specialisation. After having implemented and tested it in chess endgame domains and on real-world data, its specialisation scheme showed some issues, what leaded them to combine it to a generalisation method, what produces a system called Generalising Closed-World Specialisation(GCWS).

A key feature of CWS is its ability to augment the hypothesis language by predicate invention. GCWS is designed to learn general rules which may have one or more levels of exceptions, what is suited for chess. In addition, incremental learning coupling with theory-guided sampling allows to deal with large data sets, since the system operates at any given time with only a subset of the total problem domain.

2.1.2 Closed World Specialisation

Minimal specialisation, i.e. most general correct specialisation, can lead to theories which are not finitely axiomatisable. To avoid this problem, Bain and Muggleton used the “negation as failure” rule in the context of normal logic programs. Following [Llo87] they develop their method of CWS with the help of semantic definitions.

Closed-World Specialisation algorithm and incremental learning

The CWS transforms a normal logic program given a single negative example. In practice, a set of negative examples has to be respected. This is often the case

within an incremental learning process which includes alternate training and testing phases. Such testing may also detect uncovered positive examples in addition to covered negative examples. So CWS is incorporated in an incremental learning system with the implementation of learning from batches of examples. The batch transformations return from a possibly incorrect and incomplete program another program which will be correct and complete with respect to the examples of the test set. In practical systems, these batch transformations are not used. Other transformations for generalisation and specialisation are used in GCWS. These are implemented using a number of ILP algorithms: The generalisation method uses versions of Golem ([MF92]), and the specialisation method uses different "test-then-specialise" versions of CWS.

To avoid over-specialisation, the incremental learning of programs where the target predicate is invoked by some other predicate is not considered here.

The examples used by GCWS in the incremental learning process are selected through theory-guided sampling: Only examples found to be exceptions towards the current program are used in subsequent learning. So much larger example sets can be handled than by a "one-shot" learning system.

Discussion of Closed-World Specialisation and Generalising Closed-World Specialisation

Two recent approaches (about machine learning and logic programming) related to the CWS algorithm are discussed here.

[Wro93] proposed the use of criteria other than minimal specialisation for theory revision. Unfortunately this framework depends on a classical logic representation, in the belief of which sets are closed theories, and thus minimal specialisations may not be finitely axiomatisable.

In Logic Programming the specialisation problem may be expressed in terms of updating logic databases. For instance, [GL90] and [GL91] have presented results for provably correct updates on normalised logic programs. Deletion and insertion are carried out by update procedures which return a set of database transactions. The specialisation operation is then the deletion of an atom from the program. This method though does not guarantee minimal specialisation.

Future work should investigate in more detail the relationship between belief revision in AGM logic and other frameworks and the method of closed-world specialisation. Perhaps in practice minimal specialisation is too conservative for machine learning applications where the main focus is on theory construction.

And it may be more suited for applications where the task is to find small adjustments to a substantial theory, which is already largely correct.

2.1.3 A classifier of KRK by optimal depth of win

As mentioned before, chess endgames have highlighted knowledge representation issues for learning systems. A well studied case is the KRK endgame, with the target predicate "White-to-move position is illegal". An incremental ILP algorithm already enabled the induction of a complete and correct solution for the KRK illegality problem ([Bai91]). A harder problem is to know if a machine could learn to play the game optimally from samples of the database and simple facts about the board geometry.

KRK BTM database

Chess endgames are complex enumerable domains, what enables the construction of databases: tables of legal positions with the number of moves until a side wins assuming minimax-optimal play. Such databases provide positive and negative examples and also an oracle ([Roy86]) for testing induced rules. These databases are generated using a single iterative process using Shannon's standard backup algorithm. They can be used to play the endgame optimally using only a legal move generator and a database look-up program. As the order of an endgame database is 64^N (with N the number of pieces) without removal of redundancies, the exhaustive enumeration is impossible for endgames with more than a few pieces. Various symmetries can be used to reduce the size of the databases, so that only the canonical positions appear. Only information on Black-to-move (BTM) positions was extracted from the database, what is sufficient to play optimally with a 2-ply (i.e. 1 move) legal move generator. In the KRK database, the symmetries enable to reduce the size of the database from potentially 262144 to 28056.

Structure of an optimal classifier

The framework of this work, called classifier, handles Black-to-moves KRK legal, canonical positions and returns an integer from 0 to 16 included or a symbol denoting draw. Its structure is shown in figure 2.1. Each separate recogniser should be induced by the GCWS algorithm, and should detect if the given position is of its depth of win. By this process of elimination a position is classified by

the minimum number of moves necessary for White to win, assuming minimax-optimal play.

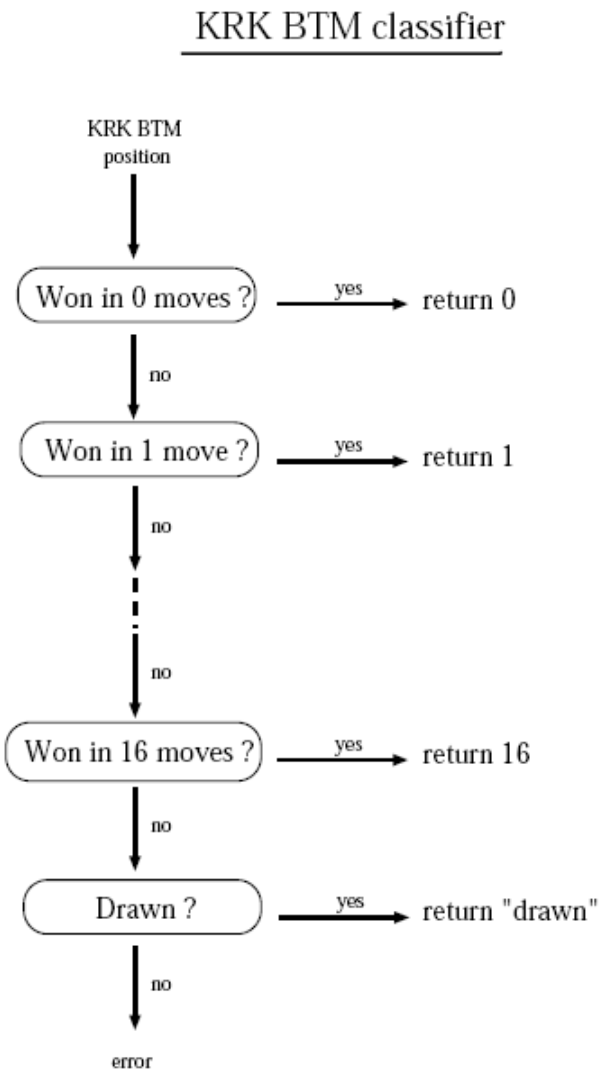


Figure 2.1: [BMS95]: Sequentially-ordered KRK BTM position recognisers from a single global classifier.

The ordering of the recognisers constraints the example sets from which each recogniser is induced and determines which exceptions have to be excluded, since the negative examples at depth D comprise only positions won at depth $> D$ plus draw. Therefore when the completeness and correctness of a recogniser is described in this paper, this is to be understood only in reference to the whole classifier structure. The classifier is implemented by employing the representation

mechanism of Quintus Prolog predicates. The predicate krk/7 with the depth of win N as first argument and the file and rank positions of the White King, the White Rook and the Black King respectively as other arguments mean that the given position can be won optimally in N moves. As Prolog indexes the clauses of a predicate in order of the first argument, this representation implements the ordering of recognisers as in figure 2.1, as long as draw positions are assigned to an integer greater than 16. The background knowledge of the algorithm was restricted to contain only one specifically chess-oriented geometrical relation, namely the symmetric difference between files and between ranks: $abs(V_1 - V_2)$ (V_1 and V_2 being the two files or the two ranks). Other background predicates available were num/1, specifying the ranks 1-8 and files a-h, and edge/1, specifying the edges of the board along rank and file[1]. These background predicates were selected as basic building blocks for the expression of piece relations in terms of geometry of the chess board. They facilitate the expression of higher-level chess concepts such as capture, safety, check, etc. Thus they supply some of the raw material for relevant predicate invention in chess domains.

Induction of complete recognisers not guaranteed correct

The learning tasks for the recognisers were created with Golem ([MF92]). For each depth of win D, the positive examples set are the positions with depth of win D, and the negative set is chosen randomly amongst the positions of depth of win $> D$. A generaliser is expected to compress, i.e. the hypothesis should be of less complexity than the original examples. Any gain in compression may be at the expense of introducing some incorrectness into the hypothesis. The method produces generalised recognisers for each depth of win, which were tested over the whole database. For large example sets (like for depth of win 14), Golem may fail to find a generalisation, what increases the complexity of the theory. The draw positions are classified by default: they are the positions which were not classified by any recogniser.

The system was tested as a compound classifier regrouping the sequentially-ordered recognisers. For each depth of win D, the tests were defined by using as a set of positive examples all the positions won at depth D and as a set of negative examples all the positions won at depth $> D$. The results are shown in figure 2.1.3. The depth 14 is perfectly classified, but very complex. The depths 16 and draw have no position, because they were classified before by another recogniser.

[1] Some other simple predicates are used to detect valid positions.

Depth	Correctly Predicted	Incorrectly Predicted	Description Complexity
0	27	99	2
1	78	433	3
2	178	178	5
3	48	418	4
4	63	1098	8
5	181	2022	9
6	67	2793	9
7	90	3427	11
8	76	1118	23
9	183	4441	23
10	221	2276	25
11	143	634	53
12	359	2517	77
13	473	1693	109
14	1153	0	4553
15	1146	410	22
16	0	0	12
draw	0	0	-
Total	4496	23560	7774

Table 2.1: [BMS95]: Sequentially-ordered BTM recognisers by depth of win. The total lack of cases for Depth=16 and draw is the result of their having erroneously been claimed by recognisers earlier in the sequence.

Induction of complete and correct recognisers

To develop complete recognisers which are also correct, a radical extension of Golem was built according to GCWS. The generalisation step is based on Golem and is called *not(Golem)* due to his ability to perform Closed-World Specialisation. The specialisation step was implemented in two variants. In the Strategy 1 specialisation, CWS modifies the current hypothesis with respect to the exception. In the Strategy 2 specialisation, the faulty clause is deleted and rebuilt with

Theory	No. of clauses	No. of invented predicates	Size of encoding in bits
Depth 0 (Strategy 1)	9	5	3491(+88935=92426)
Depth 0 (Strategy 2)	6	3	3174(+77524=80698)
Depth 0 (All Examples)	6	3	3177(+77524=80701)
Examples			278893
Depth 1 (Strategy 1)	16	7	4310(+112116=116426)
Depth 1 (Strategy 2)	11	4	3633(+96964=100597)
Depth 1 (All Examples)	12	4	3588(+100482=104070)
Examples			278608
Depth 2 (Strategy 1)	31	7	4931(+138474=143405)
Depth 2 (Strategy 2)	23	7	4381(+126438=130819)
Depth 2 (All Examples)	20	6	4329(+120802=125131)
Examples			277833
Depth 3 (Strategy 1)	61	17	7322(+164311=171633)
Depth 3 (Strategy 2)	44	12	5784(+151253=157037)
Examples			275388
Depth 4 (Strategy 1)	116	38	12608(+189444=202052)
Depth 4 (Strategy 2)	89	22	9035(+178885=187920)
Examples			274583
Depth 5 (Strategy 1)	338	88	30974(+230402=261376)
Depth 5 (Strategy 2)	185	40	15445(+20655=222000)
Examples			272614

Table 2.2: [BMS95]: Comparison of specialisation strategies.

respect to the exceptions. Each depth of win D was treated as a separate learning problem, with all the positions won at D as positive examples and randomly chosen position won at $> D$ as negative examples.

The GCWS incremental algorithm was run to induce six separate complete and correct recognisers for depths of win 0 to 5 moves with both strategies. The induced theories begin to pass the human horizon of of comprehension by depth 2. The results for both strategies are shown in figure 2.1.3. For depth 0 to 2, the results for both strategies are compared with the results of a batch learning of *not(Golem)* of all examples. The complexity is first measured in number of Prolog's clauses and the number of the invented predicates, and second in the encoding size of the recognisers. The encoding size is calculated as encoding size of the hypothesis plus the encoding size of the proof (a specification of the derivation of the examples from the hypothesis). It can be compared to the size of all the learning examples.

Some common concepts that are present in different recognisers cannot be detected with this method. So a test was made to combine the recognisers for depth 0 and 1, to see if the idea could bring better results. A common depth 0 and 1 recogniser was build and contains 13 clauses. No more tests were made to deepen this method.

Discussion

The depth 0 and depth 1 recognisers are similar to those of an earlier study ([Bai94]) and were validated as meaningful by a chess expert. At depth 2, the recogniser has 2 clauses which cover two thirds of the total positions. Moreover at this depth, exceptions to exceptions appear, what was not seen before.

As seen in figure 2.1.3, the recognisers are more compact than the examples set. But the compression is small for depth 5, and the complexity trend suggests that this approach would be bad for depth 6 and more. Strategy 2 is more compressive and produces less complex theories, so it is here more suitable than Strategy 1.

The test for a common concept in different depths could be an interesting idea to push on, because the depth 0 and 1 recogniser contains 13 clauses, compared to 17 clauses for the 2 recognisers for depth 0 and depth 1.

2.1.4 Concluding Remarks

The KRK problem is a good example of a large-scale and unstructured data problem, particularly with the aim of transparent theories. The results of the current work surpass our best previous solution, what shows the applicability of our method of incremental learning in a non-monotonic representation. However our method may fail for depth bigger than 5 in the KRK problem. It seems that further progress will depend on approaching the learning task in a different way. The decline in compression for recognisers as depth of win increases suggest that more various predicates should be used, either with user-supplied background or with other predicates invention methods. A tree could replace the sequential organisation of the recognisers, in which the depth of a position is framed more precisely at each node. The optimal criteria could be also adapted, to another metric for example.

2.1.5 Benefit for our Work

The system GCWS explained in the paper and *Golem* are based on several chess patterns used to describe the board and to test some properties of the position (like if the position is valid or not), as recalled in the paper of J. Fürnkranz: [Fue93]. These patterns are general chess patterns or simple description patterns, and they are exactly the types of attributes we need, since we aim to describe the board within the KRK ending using chess knowledge. The considered patterns are listed in the table 2.3. Other patterns were found by the system, but they are too complex to be simply used.

distance(X, Y)	returns the symmetric difference between the files or the ranks of 2 figures
adjacent(X, Y)	determines if the files or the ranks of 2 figures are side by side or the same
between(X, Y)	determines if the file (or the rank) of a figure is between the files (or the ranks) of 2 others

Table 2.3: Useful Attributes on which Bain's paper ([BMS95]) is based.

2.2 On Learning How To Play, by E. Morales ([Mor97])

In his paper, Morales describes *Pal-2*, an extension of the *Pal* ([Mor92]) learning algorithm. The both systems *Pal* and *Pal-2* learn chess patterns from traces of games. *Pal-2* also learns strategy rules defined with the help of the learned patterns. Morales has tested *Pal-2* on the KRK endgame, and the results include some learned patterns, what we are in interest in.

2.2.1 *Pal-2*

Perception and recognition of patterns seems to be a good way to play chess efficiently. The key is to detect good patterns and to learn how to combine them in rules to build a valid game strategy. *Pal-2* can both acquire patterns from board positions like *Pal* and build a playing mechanism based on learned condition-action rules using the detected patterns.

Learning rules

Pal-2 follows traces of games, and for each move made by the winning side, it tests all its known patterns before and after the move and constructs playing rules. A rule is a list of conditions before the move (in form of detected patterns), a move to do, and a list of conditions after the move. Patterns that do not change as a consequence of the move are eliminated from the conditions of the rule after the move. The positions of the pieces are replaced by variables in constructed rules. The coincidences in the values have to be eliminated later by more general rules.

For each move of the winning side, *Pal-2* compares the move made in the trace with the moves suggested by his applicable rules. If a rule suggests the good move, this rule becomes the first one in the rules' list. If no rule is applicable or suggests the good move, a new rule has to be learned and *Pal-2* asks the user for a new pattern.

The rules are ordered in the list as following: the more specific first and the more general last. Every new rule is compared with the other rules for subsumption. When the new rule is recognised as a more general form of another one, the older and specific one is deleted. If the new rule is a more specific form of an older rule,

it is deleted and the older one comes at the beginning. So the rules are ordered and the coincidences are removed.

Learning Patterns

At the beginning, *Pal-2* has only general purpose chess knowledge, it contains no playing rule and knows the only pattern "being in check". The main problem of learning a new rule is to know if a rule or a pattern is lacking. *Pal-2* cannot find it out alone, that is why the user has to choose if *Pal-2* needs a new pattern.

Patterns are learned with *Pal*. *Pal* is an inductive logic programming (ILP) system, which uses a restricted *least general generalisation* algorithm to guide its learning strategy. It has to be provided with a set of board positions containing the new pattern, and applies a new constraint which identifies the role of the pieces in the different examples to reduce the complexity of the generalisation algorithm.

primary name	our designation	description
ThreatkR	Adjacent WR-BK	The opposite king threatens our rook
Rook_divs	WR Between WK-BK	Rook divides both Kings either vertically or horizontally.
Alm_oppos	Kings almost in opp.	King is "almost" in opposition with the opposite King.
L_patt	L Pattern	The 3 pieces form a L shaped pattern with the opposite King in check by the rook.
rkK	WK Between WR-BK	The rank / file of Rook is $<$ / $>$ rank / file of King which is $<$ / $>$ rank / file of opponent's King.

Table 2.4: [Mor97]: Patterns from *Pal-2*

2.2.2 Results

In the experiments, the user followed a very simple strategy where the opponent's king was always moving away from the other king, the rook was dividing both kings eather vertically or horizontally and the opponent's king was not allowed

to move more than one file or rank away from the rook. The patterns learned by *Pal-2* are listed in table 2.4.

2.2.3 Benefit for our Work

The system described in this paper has found some patterns (listed in table 2.4) which it has learned while playing the KRK ending. It used these patterns to describe the board and find when it should apply some particular rules. Maybe the patterns it found while playing its particurlar strategy can help us describe the chess board in a more general context.

2.3 Learning Long-Term Chess Strategies From Databases, by A. Sadikov ([Sad06])

In his paper Sadikov investigates the learning of long-term strategies from computer-generated databases. The method consists in splitting the whole game in stages, and to achieve local goals in each stage. In the last stage, the goal is to mate the black king. To identify the stages, a list of known attributes was used, and a big change of their values defines the border between two stages. Sadikov experimented his approach in the KRK endgames and in the more difficult KQKR endgame.

2.3.1 Method

The work is based on chess endgames for which a complete database exists. The basic idea is to break down the endgame into stages to build a strategy with subgoals to play the endgame. In each stage, certain features are important and others are not. The stages have to be automatically extracted from the database given a set of position attributes specific for the endgame.

To detect borders between the stages, the changes of the values of the attributes are measured during the play. Sadikov assumed that large changes indicate borders between stages. So the computing of the stages starts with the calculation of the values $a_{i,j}$ of the attributes for a depth of play `i`, for all `i` in the database (ie $i \in [0, 16]$ for the KRK endgame) and for all attributes `j`. Then the information gain ratio of these values between the levels `i` and `i+1`: $g(i, j)$ are calculated, and regrouped in a vector G(i) = (g(i,1), g(i,2), ...). The assumption is that the

behaviour of G(i), when level i changes, is indicative for the stage of play: If G(i) stays about the same, then both levels belong to the same stage, otherwise they are in different stages. To test the evolution of G(i), a function Corr(i) is used:

$$Corr(i) = Corr(G(i), G(i+1)) = \frac{E(G(i) \cdot G(i+1)) - E(G(i)) \cdot E(G(i+1))}{\sqrt{Var(G(i)) \cdot Var(G(i+1))}}$$

Here, E(X) is the average value of the values in vector X and Var(X) the variance of values in X.

kdist	the distance between the two kings
rsafety	rook safety: defined as the distance between the rook and the enemy king
oppL	true if the kings are in opposition
edist	black king's distance to the closest edge
fspace	the space the black king has, as limited by the rook
rdiv	true if the rook divides the two kings
srdiv	true if the rook divides the two kings in direction towards the closest edge
grook	true if the rook holds the black king towards the closest edge
soppL	true if the kings are in opposition that forces the black king towards the closest edge
squeeze	true if both 'grook' and 'srdiv' hold
cdist	the distance of the black king from the closest edge
wkcdist	the distance of the white king from the central cross

Table 2.5: [Sad06]: Attributes from Sadikov

To segment the database, first all the G(i) are calculated from the database. Second Corr(i) is calculated from G(i). And third: Corr(i) is plotted versus i, and the local minima are candidate points for borders between stages of the play. When the database is broken down in stages, a classifier can be build, that returns the stage of play for a given position. The intention is that such classifiers for each stage of play are comprehensible, so that they can characterise the stage such that human players can follow the long-term strategy for playing this endgame.

To play the endgame, there are two possible ways: On the one hand the classifiers can be used to define subgoals for each stage (i.e. the conditions to attain the next stage). On the other hand, the classifiers can return an estimation of the distance-to-win. This estimation can be used as a heuristic evaluation function for minimax search up to a chosen depth. The move to play is decided by this minimax search. More precisely, from a given position, the classifier returns the stage of play the position was estimated to be in. Each stage of play has a particular regression function that returns for any position in this stage an estimation of the distance-to-win.

2.3.2 Application to KRK

For the KRK endgame the attributes listed in the table 2.5 were used. These attributes are based on a set used in some previous studies, for example in [Bra01].

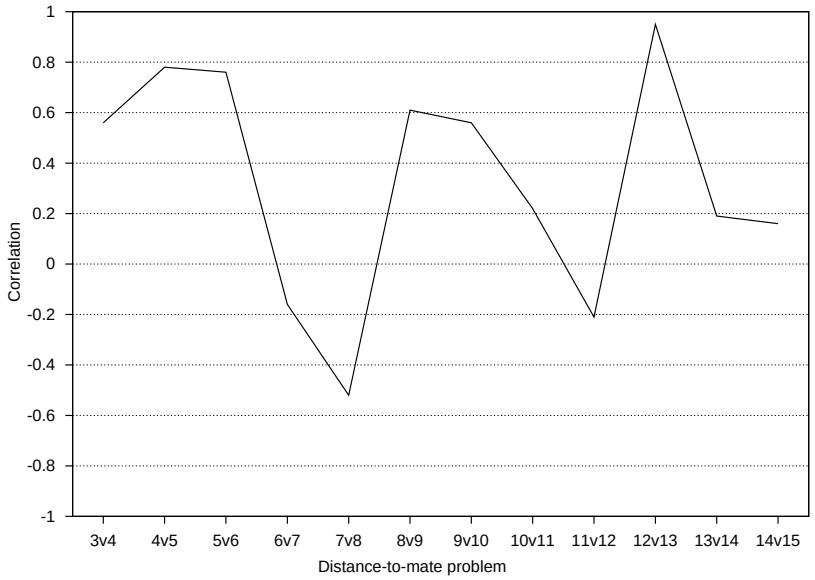


Figure 2.2: [Sad06]: Phase Separation for KRK Endgame

From this list, the value vectors can be calculated and thus the adjacent gain vectors. The figure 2.2 shows the correlation plot measuring the similarity between these vectors. There are two disctinct local minima: between levels 7 and 8 and between levels 11 and 12, so the KRK endgame was divided into three phases: Phase Close (levels 0 to 7), phase Medium (levels 8 to 11) and phase Far

(levels 12 to 16). The levels 0 to 2 don't appear on the plot because they are less numerous in the database, being so a kind of exception.

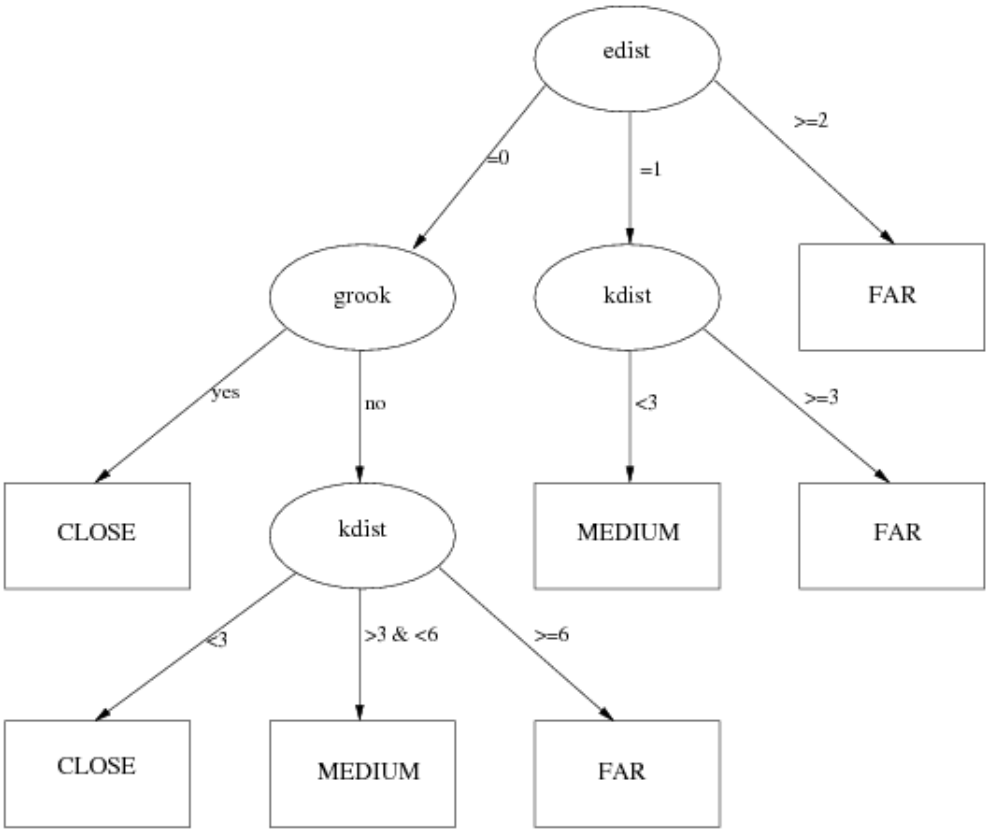


Figure 2.3: [Sad06]: Decision tree for classification into KRK phases

The induced decision tree for classification of positions into the three phases is shown in figure 2.3. It uses only three attributes and is simple to understand. The differences between the neighbored phases give the local targets to achieve next phase. Phase FAR consists mostly of positions where the black king is not on the edge of the board. The objective in this phase is thus to put the black king to an edge. Phase MEDIUM consists of positions where the black king is on the edge of the board, but the white king is far from it. The objective here is hence to bring the white king near the black one. Phase CLOSE regroups the other positions.

Given these stages, different types of evaluation functions were built: Some global functions, based on all the positions of the endgame and varying on the number of attributes they use; some local functions, based only on the positions of a particular stage and varying on the way they are built together to return a global function; and the same local functions used globally on any position. All the obtained regression functions are represented as coefficients for each used attribute.

Regression function	**Success Rate**	**Suboptimality level (in plies)**
global with full set of attributes	58.43%	+59.42
global w/o boolean attributes	100%	+13.64
global w/o boolean attributes and w/o 'wkcdist'	100%	+7.64
local with tree classifier	100%	+8.44
local with perfect classifier ("cheat")	100%	+12.96
local learned on tree classified data with tree classifier	100%	+16.78
CLOSE local used globally	100%	+8.52
MEDIUM local used globally	100%	+7.42
FAR local used globally	100%	+8.58

Table 2.6: [Sad06]: The performance of KRK evaluation functions

The calculated regression functions are used to play the endgame. The results are shown in table 2.6. The success rate tells the proportion of the positions in which a 6-ply minimax search using a given function was able to win. The suboptimality level tells how many more plies above optimal the player needs in average. The expectation was that local regression functions are better than global functions and than local functions used globally. This is surprisingly not true. The local functions used globally are the best and the global function without boolean attributes and without 'wkcdist' is also better than expected. These results indicate that the endgame decomposition into stages did not contribute to the success of play implemented as minimax with distance predictor. Until better ways of using it are found, the decomposition into stages only provided general hints about the endgame to a human player. A better comprehension of the proceeding details can be achieved by the observation of examples of plays of the player compared to the ideal player (which uses the whole database).

2.3.3 Experiments in the KQKR Endgame

Sadikov has also tested his methods on the King & Queen vs King & Rook (KQKR) endgame. As the results he obtained there are not related to our study, we will not detail them any further.

2.3.4 Benefit for our Work

The attributes Sadikov used are good patterns for the KRK endgame which are often used. They should allow us to distinguish the different depths of win, since Sadikov used them to separate the game in different phases. In table 2.7 we renamed the list shown in table 2.5 so that the names correspond to our designation.

Primary name	Our designation	Description
kdist	Distance WK-BK	the distance between the two kings
rsafety	Distance WR-BK	rook safety: defined as the distance between the rook and the enemy king
oppL	Kings in opposition	true if the kings are in opposition
edist	Dist. BK-Closest Edge	black king's distance to the closest edge
fspace	BK's free space	the space the black king has as limited by the rook
rdiv	WR Between WK-BK	true if the rook divides the two kings
srdiv	WR divides K. tow. CE	true if the rook divides the two kings in direction towards the closest edge
grook	WR holds BK tow. CE	true if the rook holds the black king towards the closest edge
soppL	Kings in opp. tow. CE	true if the kings are in opposition that forces the black king towards the closest edge
squeeze	WR squeeze BK	true if both 'grook' and 'srdiv' hold
cdist	Dist. BK-Closest Corner	the distance of the black king from the closest corner
wkcdist	Dist. WK-Central Cross	the distance of the white king from the central cross

Table 2.7: [Sad06]: Attributes from Sadikov, renamed

2.4 Leonardo Torrès y Quevedo's KRK Machine

The description of a chess machine was described first in French in [LaN14], and then in English in [Lev88]. Mr. Torrès y Quevedo, a Spanish engineer, built around 1910 a chess-playing machine that could play without help the KRK endgame against a human player. The operating mode of this machine is very interesting to investigate.

2.4.1 Operating Mode

The machine can not start from any starting position. A certain order in the positions of the pieces has to be verified at the start: A vertical direction has to be defined, and the black king should be beneath the white king and the rook. The white rook has to be either on the a-file or on the h-file.

in_zone_ac/1	tests if the rook and the black king are both in 'a', 'b' or 'c'
in_zone_fh/1	tests if the rook and the black king are both in 'f', 'g' or 'h'
vertical_distance/3	returns the vertical distance between two figures (WR and BK or WK and BK)
horizontal_distance/3	returns the horizontal distance between two figures (WK and BK)
odd/1	tests if a number is odd
even/1	tests if a number is even
zero/1	tests if a number is equal to zero
reduce_distance_by_one/3	substracts a distance by one
>/2	tests if a number is larger than another

Table 2.8: List of the attributes used by Leonardo Torrès y Quevedo's machine

If the opponent plays an illegal move, the machine detects it and refuses to play, and a light gets on. Once three such illegal moves have been made, the robot ceases to play alltogether. If on the contrary the defence plays correctly, the robot will carry out one of 6 operations, depending upon the position of the black king compared to the positions of the white king and the rook. The possible moves executed by the machine are: The rook moves to the a-file; the rook moves to the h-file; the rook moves one square down; the king moves one square down; the king moves one square to the left; the king moves one square to the right. One of

these moves is chosen by the machine with the help of a rules system, depending on the position of the black king. The 6 rules used by the machine are listed here:

If the black king is in the same zone as the rook, then the rook moves away horizontally. Otherwise look at the vertical distance between the black king and the rook. If it is more than one square, the rook moves down one square, but if it is just one square, look at the vertical distance between the two kings. If it is more than two squares, the king moves down one square, and if it is only two squares, look at the number of squares representing their horizontal distance apart. If it is zero the rook moves down one square; if it is even but not zero, the white king moves one square towards the black king; if it is odd, the rook moves one square horizontally. The predicates used to to apply these rules are summarised in table 2.8.

2.4.2 Benefit for our Work

From the patterns used by the chess machine, we extract some attributes which could help us to describe the chess board. They are listed in table 2.9.

Same Zone WR-BK
Vertical Distance WR-BK
Vertical Distance WK-BK
Horizontal Distance WK-BK

Table 2.9: List of the attributes originated from Leonardo Torrès y Quevedo's machine

Since these attributes allow the machine to play the endgame automatically, they may help us to build our chess knowledge. As they are direction dependent, we will have to find a way to adapt them for all directions so that they can give us a good information for all positions. This adaptation will be the key of our success in using the attributes, and will be described in the next chapter.

Chapter 3

Knowledge

This chapter contains the descriptions of the attributes we used to build different new databases. Our idea is that a good combination of these attributes could define an interesting knowledge that can help to compress the database without losing much information.

3.1 Preliminaries

3.1.1 Attributes Classification

The attributes come mainly from the four sources we described in chapter 2, but some other were added since they seemed important to us. They are all listed with the set they belong to in the table 3.1. The first four sets, named *Bain*, *Morales*, *Sadikov* and *Torres* in reference of the main author of the papers they come from, represent also the origin of the attributes, in contrary to the fifth set, called *My Selection*, which regroups new attributes that we found interesting and a selection of some attributes from the source papers. These five sets and a sixth one containing all attributes will be used in the next chapters to build new databases.

To keep a clear overview, we classified the patterns in 7 categories depending on their nature: Some attributes just refer to figures distances, some other on the position on the board... All the attributes from each category will be explained with examples. Their properties will be detailed as well as the reasons why they are interesting and in which situation. Moreover the implementation of the attributes in the script will be also described.

Name	Bain	Morales	Sadikov	Torres	My Selection
Figure Distance Patterns					
1. Distance WK-BK			×		×
2. Distance WR-BK			×	×	×
3. Distance WK-WR					×
Figure File and Rank Distance Patterns					
4. File distance WR-BK	×				
5. Rank distance WR-BK	×				
6. File distance WK-WR	×				
7. Rank distance WK-WR	×				
8. File distance WK-BK	×				
9. Rank distance WK-BK	×				
10. Alignment dist. WR BK					×
Adjacent Figure Patterns					
11. Adjacent WK-WR	×				
12. Adjacent WK-BK	×				
13. Adjacent WR-BK	×	×			
Between Figure Patterns					
14. WR Between WK & BK	×	×	×		
15. WK Between WR & BK	×	×			
16. BK Between WK & WR	×				
Board Distance Patterns					
17. Dist. BK-Closest Edge			×		
18. Dist. BK-Closest Corner			×		×
19. Dist. WK-Central Cross			×		×
20. Dist. WK-Closest Corner					×
Orientation-based Patterns					
21. Vertical distance WR-BK				×	
22. Vertical distance WK-BK				×	
23. Horizontal dist. WK-BK				×	
24. Same Zone WR-BK				×	
Figure Relation Patterns					
25. Kings in opposition			×		
26. Kings almost in opp.		×			×
27. L pattern		×			
Figure Relation Patterns Associated with the Board					
28. WR divides K tow. CE			×		
29. WR holds BK tow. CE			×		
30. Kings in opp. tow. CE			×		×
31. WR squeeze BK			×		
32. BK's free space			×		×
33. BK's available squares					×

Table 3.1: Attributes with their category and the set their belong to

3.1.2 Basis Concepts

Figures Distance

The notion of distance that is used here is particular for the chess board: the distance between two figures is the maximal value of the difference between the files of both figures and of the difference of their ranks. In other words, it is the smallest number of moves a king would need to go from the place of the first figure to the place of the other figure on a free board (i.e. without any interference with other figures). For example on the figure 3.1, the white king is in a2 and the black king in c5 so their distance is 3.

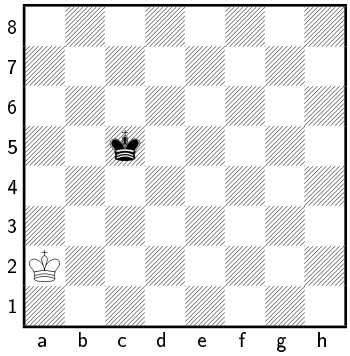


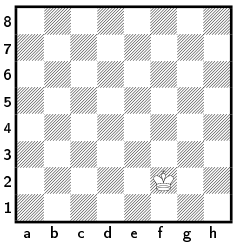
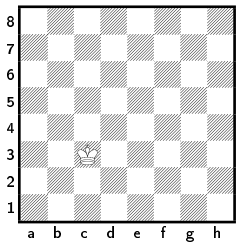
Figure 3.1: Distance example on the chess board: $distance = 3$

Closest Edge

Some attributes are based on the relation between a figure and the border of the board. In this case we speak of the closest edge or the closest corner of the figure. In some cases there may be a problem to define which edge is the closest. The definition is done with a simple method: The distance from the figure to each edge is calculated (the edges are simply the 2 files a and h and the two ranks 1 and 8). Then we compare these distances with each other. If one distance is strictly smaller than the 3 other, like on figure 3.2 first board, then the closest edge is the coresponding one. If there are 2 distances which are the smallest, like on figure 3.2 second board, there are two closest edges, i.e. each one of the two corresponding edges can be taken as the closest edge. We then have to pay attention while implementing the attributes using this feature to determine which one of the two closest edges should be used. Except the attribute *Distance to the closest edge*, for which the value is the same even if there are two closest edges,

only boolean attributes use this feature. For such kind of attributes, if there are 2 closest edges, the returned value is `true` if the test is `true` for at least one of the two edges.

Some attributes use similarly the concept of closest corner. Such a corner is determined with the same method, but there can be only one closest corner.

Closest Edge:	'1'	'a' and '1'
Distance to CE:	1	2
Closest Corner:	'h1'	'a1'
Distance to CO:	2	2
Distance to CR:	2	1

Figure 3.2: Examples for the concepts *Closest Edge* (CE), *Closest Corner* (CO) and *Central Cross* (CR)

Notations

In this chapter, we use the standard notation *WK* for the white king, *BK* for the black king and *WR* for the white rook. Some other shortcuts may also appear, like *CE* for *Closest Edge*, *CO* for *Closest Corner* or *CR* for *Central Cross*.

3.2 Figure Distance Patterns

3.2.1 Distance WK BK

How far from each other are the two kings?

This attribute characterises the distance between the two kings. It can have any integer value between 2 and 7 (because the positions in the database are all legal, so a distance of 1 is not possible). For the winning side, a small value for this attribute is preferable, so that the enemy king is prevented to move as it wants

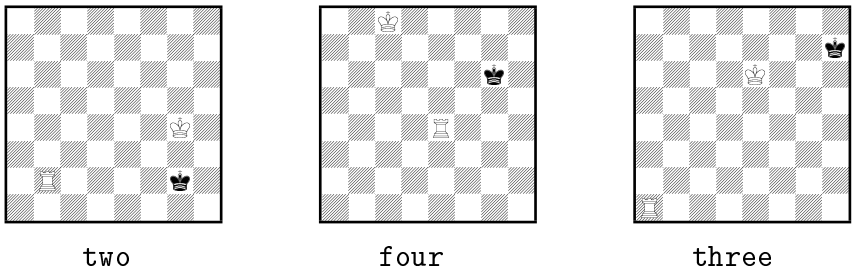


Figure 3.3: Examples for the Attribute "Distance WK BK"

to. For example on the first board of the figure 3.3, the attribute is equal to 2 and the black king is forced to move one square down to the edge. In this case, this is also due to other reasons, but in general a large king distance (larger than 5 typically) means a long game (a distance to win of 6 or more, except the draws). But a small king distance does not imply anything on the depth of win.

3.2.2 Distance WR BK

How far is the white rook from the black king?

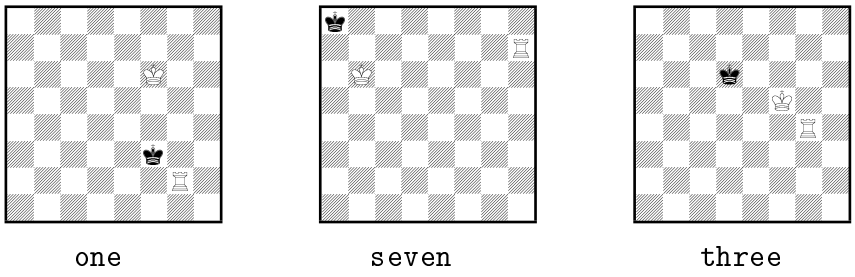


Figure 3.4: Examples for the Attribute "Distance WR BK"

This attribute contains the distance on the board between the black king and the white rook. Its possible values are all the integers between 1 and 7. A value of 1 means that the attribute *Adjacent WR BK* is `true`. A small value here is dangerous for the white side, because the rook may be taken by the black king, and the game is then a draw. To avoid it, the white side has to react and to modify his strategy. Since the distance on the board makes no difference between file distance and rook distance, this attributes give no indication if the rook is on

a good position to hold the black king toward an edge or a corner. For example on figure 3.4, on the board in the middle, the white rook is far from the black king but is in a good position: The black side can be mated in 1 move.

3.2.3 Distance WK WR

How far are the white king and the rook from each other?

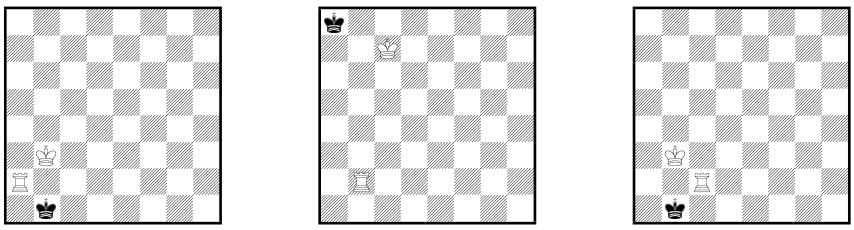


Figure 3.5: Examples for the Attribute "Distance WK WR"

This attribute contains the distance on the board between the white king and the rook. So it can be called the protection distance. Its possible values are all the integers between 1 and 7. A small value here can be good for the white side, because then the rook may be protected by the king. But in the standard strategy this attribute is not particularly small at the end, because the rook can act from the other end of the file or the rank. As the two kings are often near from each other, the rook should better be far from both. This attribute was not used in the source papers, but it seemed interesting to me to test it because of his similarity to the two others of this subsection: *Distance WK BK* and *Distance WR BK*. On figure 3.5, the value of the attribute is one for the first and the third board, but the distance to win is 7 moves in the first case and only 1 in the second case. And for the board in the middle the distance WK - WR is five and the distance to win is one as well. So this attribute alone does not tell anything about the advancement of the game.

3.3 Figure File and Rank Distance Patterns

In this section, the file and rank distances between the figures are explained. The file distance between two figures is calculated by doing a simply difference between

the file of the first figure and the file of the second one. It is the same for the ranks. So the third figure is not taken into account for the calculation. Because of the symmetries of the board and the figures' moves, the rank and file distances have the same meaning taken separately, so only the file distance attributes will be explained, knowing that the rank distances have the same properties. The possible values for these seven attributes are the integers between 0 and 7.

3.3.1 File Distance WR - BK

How far from each other are the file of the white rook and the file of the black king?

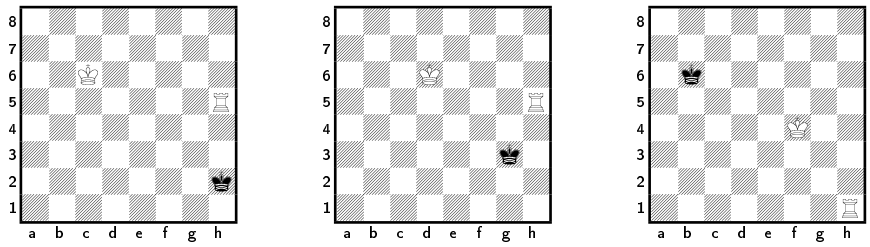


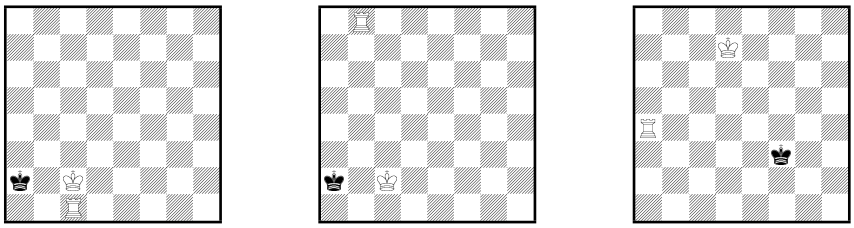
Figure 3.6: Examples for the Attribute "File Distance WR - BK"

This attribute calculates the distance between the file of the white rook and the file of the black king. A value of zero means that the black king is in check, like on the first board of figure 3.6. If the value is 1, then the rook stands next to the black king, limiting thus its move possibilities. The other values are less interesting, they just tell how many place is left for the black king in the corresponding direction.

3.3.2 File Distance WK - WR

How far from each other are the file of the white king and the one of the rook?

This attribute gives the distance between the file of the white king and the file of the rook. If its value is 0, then the white king and the rook are on the same file, what can limit the move possibilities of the rook, like on the first board of figure 3.7 where the distance to win is 7 although the black king is near a corner and in opposition with the white king. A value of 1 can be good if the distance between the kings is not too large, like on the second board.

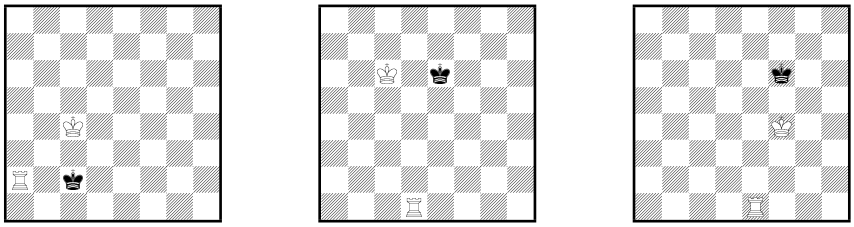


$d = 0$ $d = 1$ $d = 2$

Figure 3.7: Examples for the Attribute "File Distance WK - WR"

3.3.3 File Distance WK - BK

How far from each other are the file of the white king and the file of the black king?



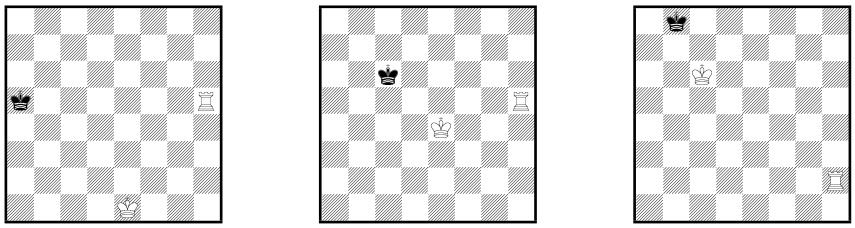
$d = 0$ $d = 2$ $d = 0$

Figure 3.8: Examples for the Attribute "File Distance WK - BK"

This attribute tells us if the two kings are on nearby files or not. It is not as meaningful as the two previous attributes, because none of the kings can move far on a file in contrary to a rook. So the attribute *Distance WK - BK* is more important, but this one was added for completeness.

3.3.4 Alignment Distance WR BK

This attribute is different from the other of the section. It also deals with file and rank distance, but here the smaller one of the file and rank distance between the black king and the rook is calculated. It corresponds to the closest distance of the rook's line to the black king. It is more interesting than the other attributes, because it gives an information which does not depend on the direction we look



$d = 0$ $d = 1$ $d = 6$

Figure 3.9: Examples for the Attribute "Alignment Distance BK WR"

from. The possible values still are integers between 0 and 7. A value of 0 still means that the black king is in check. But a value of 1 is quite interesting, as it means that the black king is hold on one side by the rook, while being at least one square far from the rook in the other direction. As a matter of fact, this attribute selects what seems to be the most interesting direction between the files and the ranks and returns the distance along the chosen direction. Since the rook can act from far away (almost the other side of the board), this attribute gives more information than the *Distance WR - BK*.

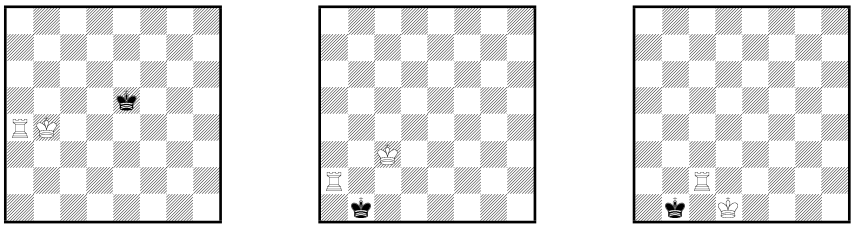
3.4 Adjacent Figure Patterns

The attributes of this section test if two of the three figures are on adjacent squares. The possible values are `true` and `false`. This property is important, because the kings can move one square in any direction. When such an attribute is true, it means that the two figures are able to interact (because two of the three figures of the endgame are kings). The importance and the consequences of the interaction depend on the nature of the two figures (mainly if they are on the same side or not). The values of these attributes are computed by testing if the distance between the two figures is one.

3.4.1 Adjacent WK WR

Are the rook and the white king on adjacent squares?

This attribute tests if the rook is protected by the white king or not. If true, the black king cannot take the rook, even if it stands next to it. In figure 3.10, the white king protects the rook on the first board, although it is not threatened. On



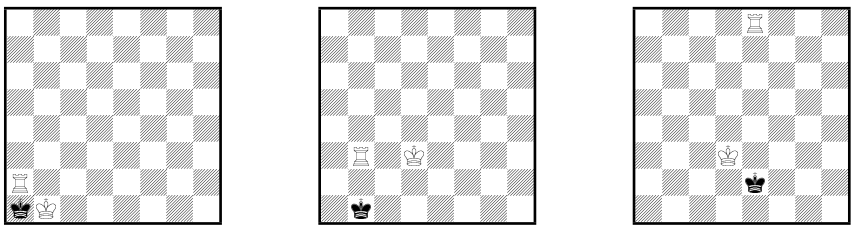
true false true

Figure 3.10: Examples for the Attribute “Adjacent WK WR”

the second board the attribute is false, and the rook is threatened, so it can be taken by the black king, and the game will thus be draw. On the third board, the rook is threatened but the attribute is true, so it cannot be taken.

3.4.2 Adjacent WK BK

Are the two kings on adjacent squares?



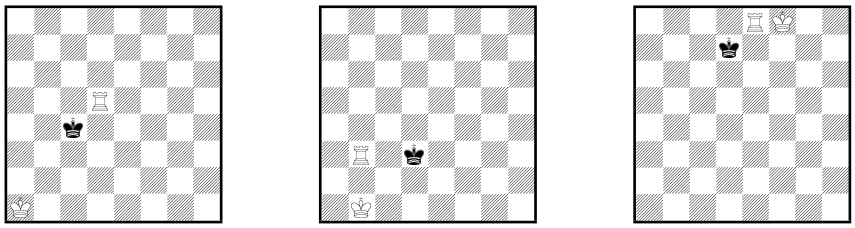
true false true

Figure 3.11: Examples for the Attribute “Adjacent WK BK”

This attribute tests if the distance between the two kings is one or more. If it is true, then the position is illegal. In fact, this attribute was used in [BMS95] to define and detect illegal positions. In our case, this attribute is not of any use if the database is built correctly, what was the case in the database we used. We only included this attribute for completeness.

3.4.3 Adjacent WR BK

Are the rook and the black king on adjacent squares?



`true` `false` `true`

Figure 3.12: Examples for the Attribute "Adjacent WR BK"

This attribute tests if the distance between the white rook and the black king is one, or more. If the value is true, then it means that the black king threatens the rook. It is an important attribute, because then the white rook may be taken by the black king, if black has the move, what was the case in the database we used, unless the white king is protecting the rook. And if the rook is taken, then only the two kings stay on the board and the game is draw! This is the case in figure 3.12 first board. On the second board, the attribute is false so the rook cannot be taken. In order to know the gravity of the value, i.e. to know if the game will be drawn or not, we have to take care simultaneously of the value of the attribute *Adjacent WK WR*. On the third board, the white king protects the rook so that it cannot be taken although the attribute is true.

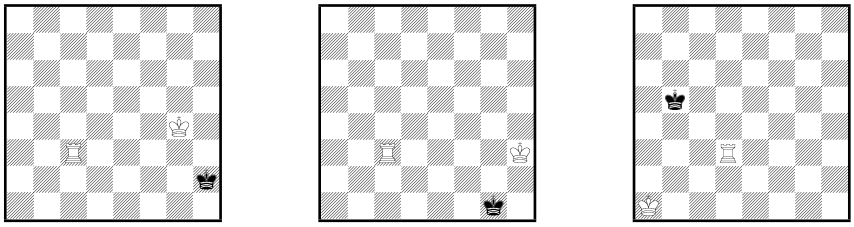
3.5 Between Figure Patterns

In this section we have grouped three attributes of the same kind: *WR between WK and BK*, *WK between WR and BK* and *BK between WK and WR*. These three attributes test if one of the three figures can be considered as standing between the two others on the board. The sense of "between" here is either concerning the ranks or the files. That is to say that a figure is between two others if its rank is between the ranks of the others *or* if its file is between the files of the others. This definition is compatible with the symmetries and rotations of the board and the figures possible moves. These attributes can take the values `true` or `false`. They are all implemented the same way: The four following possibilities are tested (the middle figure is the one which should be between the two others). Is the file of the middle figure larger than the file of the first other and smaller than the file of the second other figure. Is the rank of the middle figure larger than the rank of the first other and smaller than the rank of the second other.

Is the file of the middle figure smaller than the file of the first other and larger than the file of the second other. Is the rank of the middle figure smaller than the rank of the first other and larger than the rank of the second other. If one of the possibilities is true, then the attribute is true, otherwise false.

3.5.1 WR Between WK and BK

Does the rook divide the two kings?



`true` `false` `true`

Figure 3.13: Examples for the Attribute "WR Between WK and BK"

This attribute tells if the rook is between the two kings. In the standard endgame strategy, it has to be `true` before the two kings get in opposition, like on the first board of figure 3.13 where the depth of win is 5. Then the rook can go "down" one square to force the black king to go towards an edge (or to mate if the black king is already in an edge). On the second board the position is slightly different and the attribute is no more true, thus the depth of win is 11. But taken alone in a general position on the board, this attribute does not give very significant information, like on the third board, where the attribute is true but the distance to mate is 13.

3.5.2 WK Between WR and BK

Is the white king between the rook and the black king?

This attribute tests if the white king separates the rook from the black king. In the standard strategy, it should happen to be `true` when the rook divides the two kings so that the black king cannot approach the rook without getting in opposition with the white king, in which case the rook can move "down" to force the king to the edge, like on the first board of figure 3.14: If the black king goes

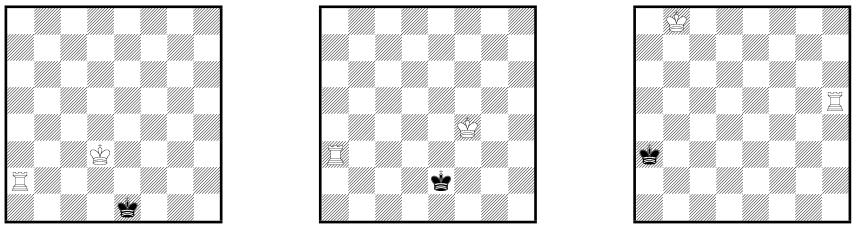


Figure 3.14: Examples for the Attribute "WK Between WR and BK"

nearer to the white rook, it gets in opposition with the white king and can be mated. On the second board the position is almost the same, but the attribute is false. The black king can then threaten the white rook without getting in opposition to the white king, so to avoid this the rook has to move. Like the previous attribute and as we can see on the third board, this attribute does not mean much taken alone in a general position.

3.5.3 BK Between WK and WR

Is the black king between the white king and the rook?

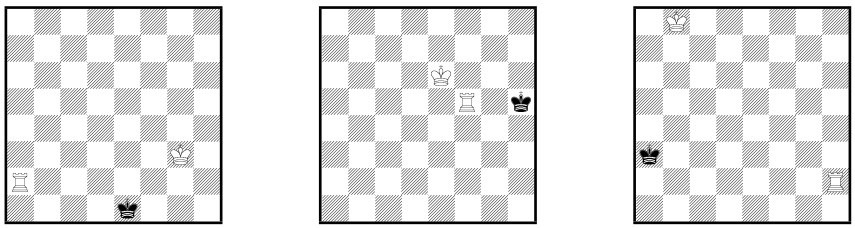


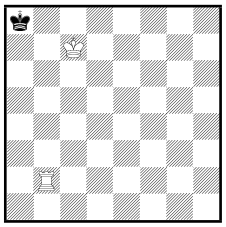
Figure 3.15: Examples for the Attribute "BK Between WK and WR"

This attribute tests if the black king stands between the white king and the rook. Such a pattern is not a target pattern in the standard strategy, but it can show that the white side is not in a optimal position, because the white figures do not force the black king to an edge. On the first board of figure 3.15, the position is like the one of the second board of the figure 3.14, so in this case this attribute can be taken as the contrary of *WK Between WR and BK*. But in general, looking at this attribute isolated does not give us any useful information.

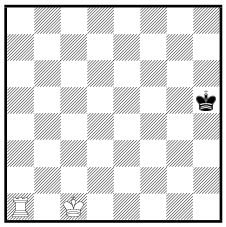
3.6 Board Distance Patterns

3.6.1 Distance BK - Closest Edge

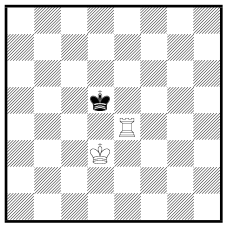
How far is the black king from the closest edge?



$d = 0, DoW = 1$



$d = 0, DoW = 14$



$d = 3, DoW = 10$

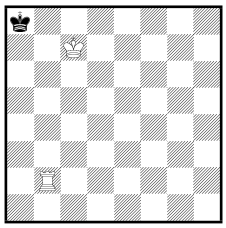
Figure 3.16: Examples for the Attribute "Distance BK Closest Edge" with the Depth of Win (DoW)

This attribute tells us if the black king is near an edge or not. Its possible values are the integers 0, 1, 2 and 3. It is important since it can give an idea of the number of moves left until the end of the game. If the black king is far away from the edge, it cannot be mated quickly: All the positions for which the attribute has a value of three have their distance to win larger than or equal to ten (except the draws). But the black king may be near an edge without having the game near its end. For example on the first board of the figure 3.16, the attribute has a value of zero and the distance to win is one: We have a typical case of small attribute value and small distance to win; but on the second board, the value of the attribute is also zero and the distance to win fourteen! On the third board, the attribute has a value of three with the smallest depth of win possible, i.e. ten. The value is computed like that: The distance between the king and each edge is calculated, and the the smallest value is returned. If the black king is on a diagonal, there are two closest edges.

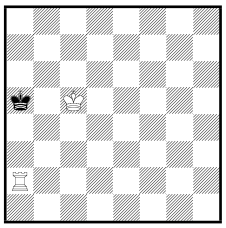
3.6.2 Distance BK - Closest Corner

How far is the black king from the closest corner?

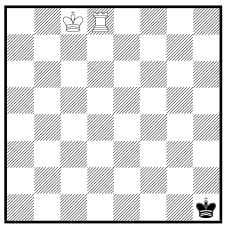
This attribute evaluates the distance from the black king to the closest corner. It is useful because it is simplier to mate when the black king is in a corner. The possible values of the attribute are all the integers between 0 and 3. Nevertheless



$d = 0, DoW = 1$



$d = 3, DoW = 0$



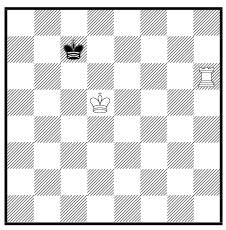
$d = 0, DoW = 12$

Figure 3.17: Examples for the Attribute "Distance BK - Closest Corner"

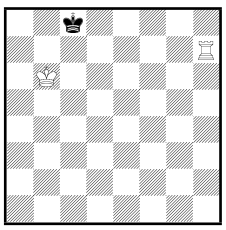
this attribute alone tells us less about the advancement of the game than the previous one (*Distance BK - Closest Edge*), since the black king can be mated in the middle of an edge, i.e. without being near to a corner (like on the second board of figure 3.17). And as well the game may still last long even if the black king is near a corner (like on the third board of figure 3.17 where the distance to win is twelve). The value of the attribute is computed that way: The distance from the black king to each corner is calculated and the smallest value is chosen.

3.6.3 Distance WK - Central Cross

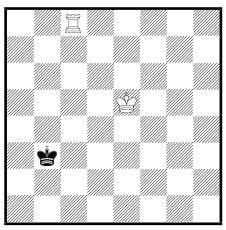
How far is the white king from the middle of the board?



`zero`



`two`



`zero`

Figure 3.18: Examples for the Attribute "Distance WK - Central Cross"

This attribute cares about the position of the white king. It tells us how far is the white king from the closest of the four fields d4, d5, e4, e5. Any integer between 0 and 3 is valid value. This attribute is the opposite of the distance to an edge: The sum of the value of this attribute and of the distance between the white king and the closest edge is always three. It does not give a direct hint on

the distance to win, since the position of the black king can be anything, but rather an indication on the state of the endgame: When the white king is in the middle of the board (i.e. when the attribute has a value of zero), it is in a good position to start forcing the black king to an edge. For example on the first board of figure 3.18, the white king is on the central cross with the smallest distance to win possible three (except the draws). On the second board, it is also on the central cross but with the largest distance to win possible, twelve. In contrary, when the white king is far from the central cross, the attribute does not give us any information about the state of the endgame.

3.6.4 Distance WK - Closest Corner

How far is the white king from the closest corner?

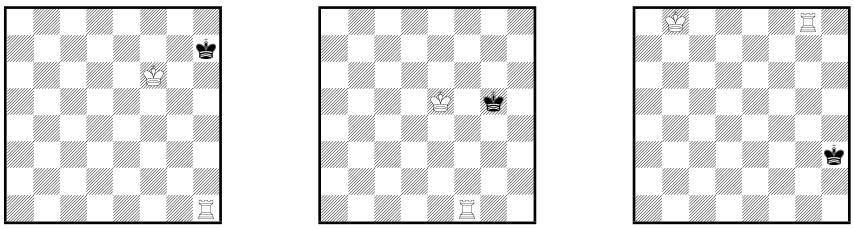


Figure 3.19: Examples for the Attribute "Distance WK - Closest Corner"

In completion to the previous one, this attribute also deals with the position of the white king on the board. It tells us how far the white king is from the closest corner. Any integer between 0 and 3 is a valid value. Since it is easier to mate the black king in a corner, this attribute is pertinent because it brings more information than the distance to the central cross (or to the closest edge).

3.7 Orientation-based Patterns

3.7.1 Vertical Distance WR - BK

How far from each other are the white rook and the black king in the "vertical" direction?

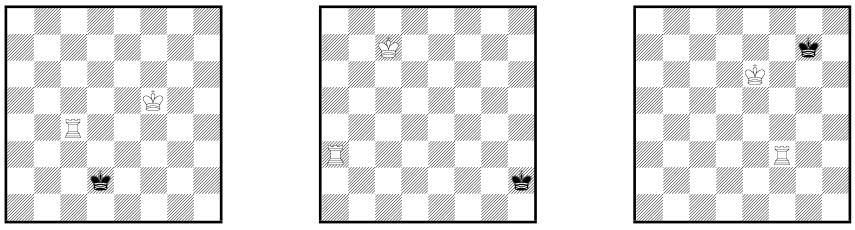


Figure 3.20: Examples for the Attribute "Vertical Distance WR - BK"

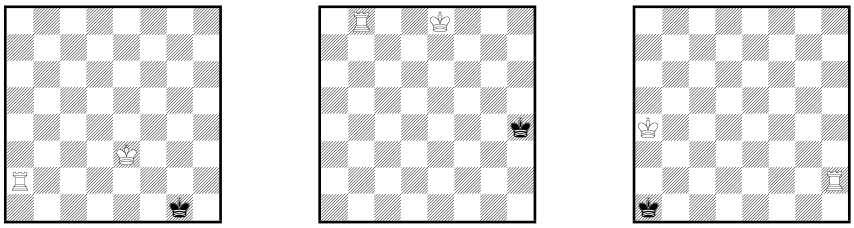
This attribute tells us how much liberty the black king has, as let free by the rook. Its possible values are `more_than_one_square` and `one_square`. If it equals one_square, then the rook is just "above" the black king and this one cannot goes upwards. We are therefore in the same situation as in the simple strategy of Morales with the black king been held against the bottom edge. The problem here is to define the vertical direction so that the attribute has a reliable value. This attribute makes no sense if the vertical direction is defined so that the rook is on the same grade as the black king. The best definition of the verticality is for us: The black king should be as low as possible, i.e. the edge representing the bottom is the edge closest to the black king. If it is not enough to define one direction, then the white king has to be as high as possible, i.e. the edge representing the bottom is the one furthest from the white king. And if there still exist two possibilities, then one of the two can be chosen randomly.

This attribute was used by the machine of Leonardo Torrès y Quevedo: If its value is `more_than_one_square`, then the machine moved the rook one square towards the black king.

3.7.2 Vertical Distance WK - BK

How far from each other are the white king and the black king in the "vertical" direction?

This attribute tells us how much liberty the white king let to the black king. Its possible values are `more_than_two_squares` and `two_squares`. The aim of this algorithm is to detect if the white king is as near as it should be to the black king and to bring it nearer if not. The role of the white king is to block the black king in a direction (the here so called upper direction). As for the attribute *Vertical distance WR - BK*, the problem is to define what is vertical, but with



two three zero

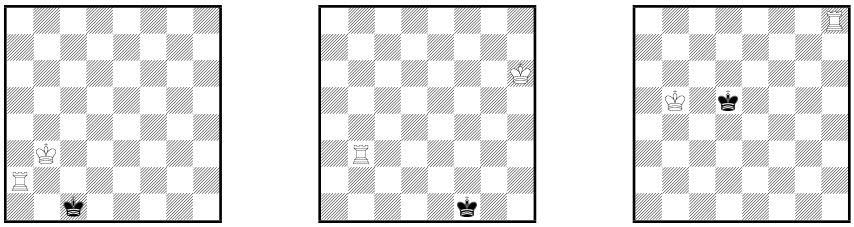
Figure 3.21: Examples for the Attribute "Vertical distance WK - BK"

the definition given for the last attribute it is well defined.

This attribute was used by the machine of Leonardo Torrès y Quevedo. If the value is `more than two square`, the machine moved the white king one square down.

3.7.3 Horizontal Distance WK - BK

How far from each other are the white king and the black king in the "horizontal" direction?



odd even zero

Figure 3.22: Examples for the Attribute "Horizontal Distance WK - BK"

This attribute compares the position of the white king with the position of the black king. Its possible values are `odd`, `even` and `zero`. It should help detecting the right move to do to bring the two kings in opposition so that the black king has to go "down" one square.

This attribute was used by the machine of Leonardo Torrès y Quevedo. If the value was `odd`, the machine moved the rook one square horizontally. If the value

was `even`, the machine moved the white king one square horizontally towards the black king. If the value was `zero`, the machine moved the rook down one square.

3.7.4 Same Zone WR BK

Are the white rook and the black king in the same zone?

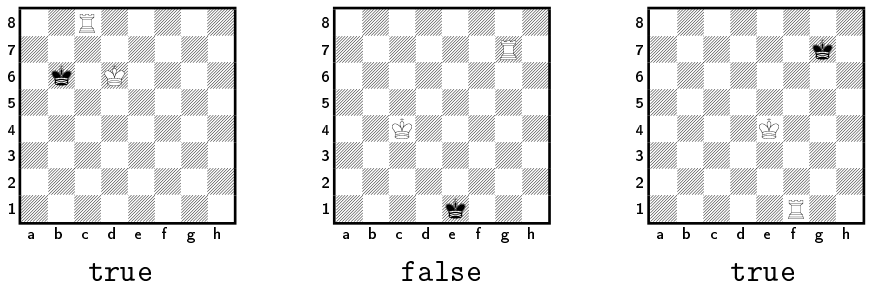


Figure 3.23: Examples for the Attribute "Same Zone WR - BK"

This attribute was used by the machine of Leonardo Torrès y Quevedo to check if the white rook could come near the black king if it moves down on the board. The possible values are `true` and `false`. You need to have defined the vertical and horizontal directions to use this attribute. Its value is computed by testing if the rook and the black king are in neighbour columns (where columns are vertical). It is `true` if the both figures are in the three first columns, or if they both are between the third and the sixth column, or in the three last columns.

You can see three examples of this attribute in figure 3.23. On the first board, the nearest edge from the black king is on the left, so the vertical areas are in fact horizontal. As the rook is on rank 8 and the black king on rank 6, they are in the same area: The attribute is `true`.

In the second example, the nearest edge is on the bottom of the board. The files e and g of the black king and the rook respectively are not in the same area: The attribute is thus `false`.

In the third example, the nearest edge is the upper one (because the black king is as far from the upper one as from the right one, but the white king is further from the upper one), so the vertical zones are the files. That is why the rook and the black king are considered in the same area: The attribute is here `true`.

3.8 Figure Relation Patterns

3.8.1 Kings in Opposition

Are the both kings in opposition?

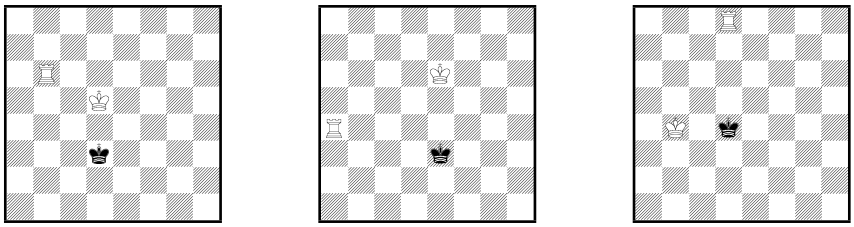


Figure 3.24: Examples for the Attribute "Kings in opposition"

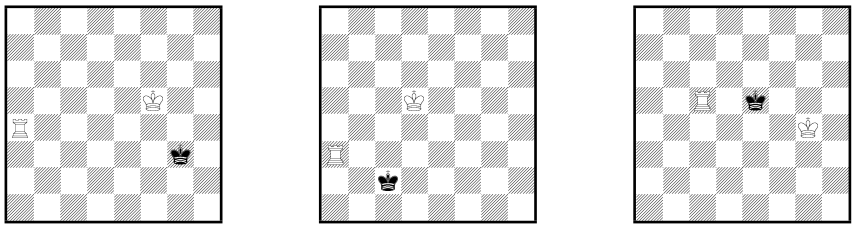
This attribute detects if the both kings are in a position on the board such that the next king that moves has to move away from the other one. That happens when the two kings are on the same file or the same rank and are separated by just one free field. The attribute can take the values `true` or `false`. It is true in one of the following two cases: The two kings are on the same file and their rank distance is two, or the two kings are on the same rank and their file distance is two. In the standard strategy to play the endgame, this attribute tells when you can move the rook to put the black king in check so that it has to move away from the white king.

Some example positions with the values of this attribute are exposed in figure 3.24.

3.8.2 Kings Almost in Opposition

Are the both kings almost in opposition?

This attribute is related to the previous one. If the both kings are not in opposition but only "almost" in opposition, they are on the right way to get in opposition. They are called almost in opposition if they are just one move away from being in opposition without being on the same file or on the same rank. The possible values of this attribute are `true` or `false`. It is true if the file distance between the both king is one and the rank distance is two, or if the file distance is two and



`true` `false` `true`

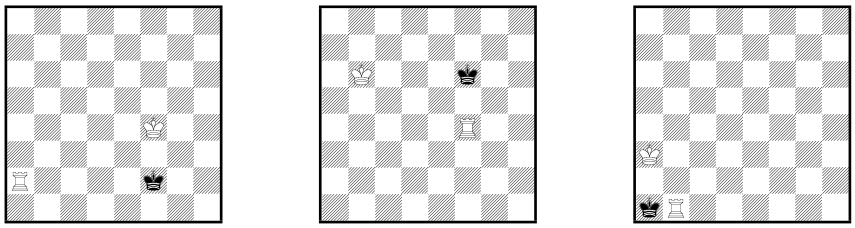
Figure 3.25: Examples for the Attribute "Kings almost in opposition"

the rank distance one. It can be summarised by: The product of the file and the rank distance is two. To obtain that the kings are in opposition, this attribute has to be first fulfilled.

Some example positions with the values of this attribute are exposed in figure 3.25.

3.8.3 L Pattern

Do the three figures on the board form a L shaped pattern with the black king in check by the rook?



`true` `false` `true`

Figure 3.26: Examples for the Attribute "L Pattern"

This attribute characterises the next step after having the two kings in opposition in the standard strategy. The rook goes "down" (i.e. in the direction of the black king) to check the black king, and as the white king blocks a whole side of the possible moves of the black king, this one also has to move "down". The possible values of this attribute are `true` or `false`. It is true in one of the following cases: The two kings are on the same file with a rank distance of two and the rook is

on the same rank as the black king, or the two kings are on the same rank with a file distance of two and the rook is on the same file as the black king.

3.9 Figure Relation Patterns Associated with the Board

3.9.1 WR Divides Kings towards the Closest Edge

Does the white rook divide the two kings in direction towards the closest edge?

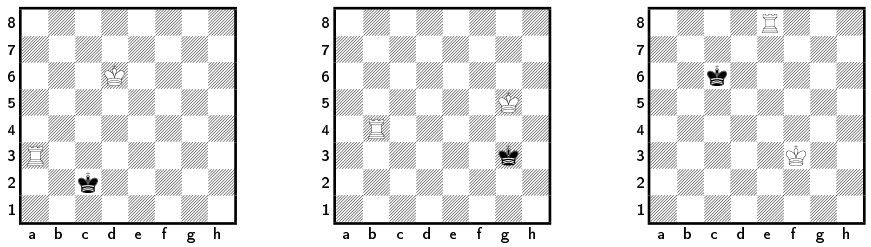


Figure 3.27: Examples for the Attribute "WR divides Kings towards the closest edge"

This attribute tells us if the white rook separates the two kings in a judicious manner. The possible values are `true` or `false`. The value is `true` if the rook is between the two kings, so that the black king is on the side of its closest edge. If there are two closest edge, any of the two can do.

In the first example of figure 3.27, the closest edge is the lower one. The rook on rank 3 is then between the white king on rank 6 and the black one on rank 1, so that the black king is on the side of the edge closest to him: The attribute value is `true`. In the second example, the rook separates the two kings, but not towards the closest edge, because the closest edge here is the right one: The attribute value is thus `false`. In the third example, we cannot determine the closest edge (because the two kings are on the diagonal a8 - h1). As the rook divides the two kings towards the left edge and the left edge is one of the closest edge, then the attribute value is `true`.

3.9.2 WR Holds BK towards the Closest Edge

Does the white rook prevent the black king from going away from the closest edge?

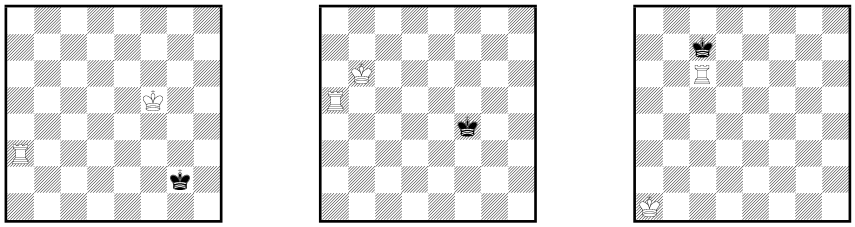


Figure 3.28: Examples for the Attribute "WR Holds BK towards the closest edge"

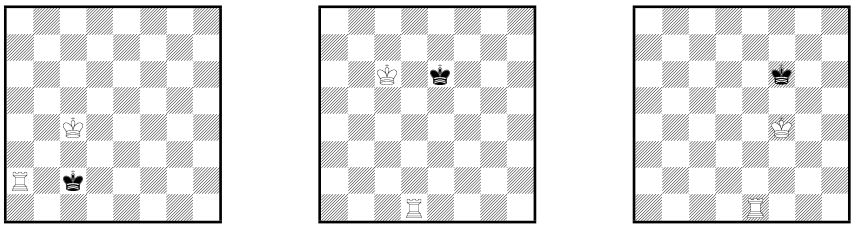
This attribute characterises the position of the rook compared to the position of the black king. It has to be just a field beyond the black king as seen from the BK's closest edge to be classified as good by this attribute. The possible values are `true` or `false`. To compute the value, the BK's closest edge is detected. If the rook's distance to this edge is just one point larger than the BK's distance, then the attribute is true. If there are 2 closest edges, the rook has to fulfil the condition for only one of the two edges.

3.9.3 Kings in Opposition towards the Closest Edge

Are the both kings in opposition, so that the black king is hold towards the closest edge?

This attribute is based on the one called Kings in Opposition, with an additional condition: the black king has to stand near the closest edge. The possible values of this attribute are `true` or `false`. It is `true` if the two kings are in opposition (same file and two ranks away or same rank and two files away from each other) and if the edge behind the black king when he "looks" at the white king is a or the closest edge.

Some example positions with the values of this attribute are exposed in figure 3.29. In the first example, we are in the standard strategy: The black king is threaten by the rook and cannot go upwards because it is in opposition with the white king, so it has to go to the bottom edge. In the second example, the kings are in opposition but not towards the closest edge, which is the upper one. In



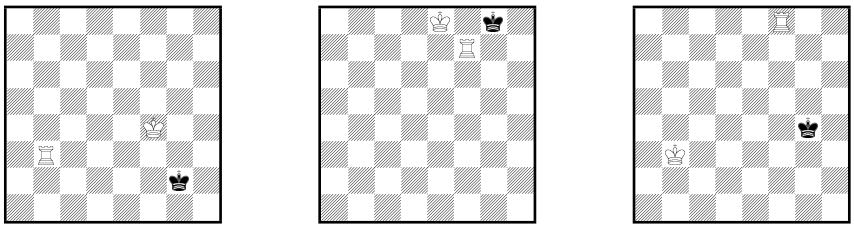
`true` `false` `true`

Figure 3.29: Examples for the Attribute "Kings in Opposition towards the closest edge"

the third example, the closest edge is also the upper one, so the attribute value is `true`.

3.9.4 WR Squeeze BK

Does the white rook "squeeze" the black king against the closest edge?



`true` `false` `true`

Figure 3.30: Examples for the Attribute "WR Squeeze BK"

This attribute combines the two following patterns: *WR divides Kings towards the Closest Edge* and *WR holds BK towards the Closest Edge*. If these two attributes are true, then this one is true as well. The possible values are `true` or `false`. The value is computed as a simple logical `AND` between the two attributes. It means that this attribute is true if the rook divides the two kings in the direction towards the closest edge, so that the black king is held towards that edge. On the first board of the figure 3.30, we have a typical position from the standard strategy where the attribute is true because the whites are going to force the black king the lower edge. On the second board, it is false because the closest

edge is the upper one, and not the one on the right. In the third example, it is true although the white king is quite far from the black king.

3.9.5 BK's Free Space

How many fields on the board are left free to the black king by the white rook?

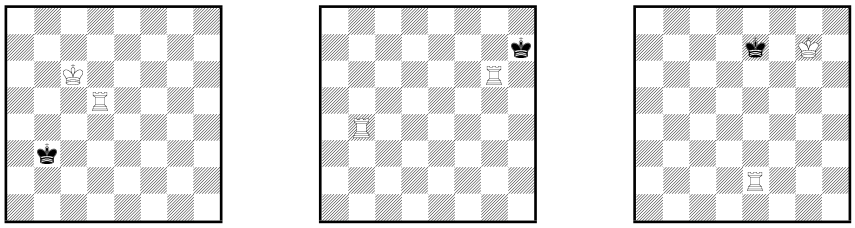


Figure 3.31: Examples for the Attribute "BK Free Space"

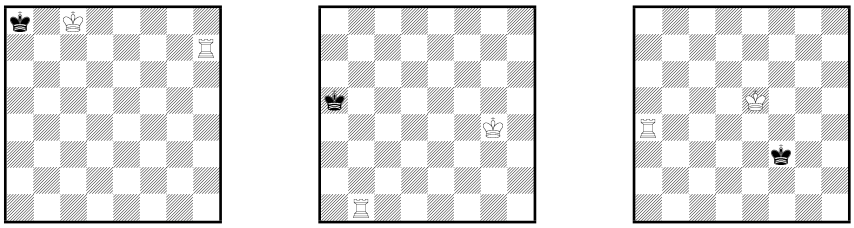
This attribute characterises the number of free squares on the board where the black king can go without being in check by the rook. The position of the white king plays no role here: Only the rook's and the black king's position are taken into account. This attribute can take some integer values (but not all) between 1 and 49. Theoretically the value could be zero, if the black king is mate, but the rook alone cannot achieve that. If the black king is on the same file or rank as the rook, the side with the biggest free space is used to calculate the attribute. Another interesting related attribute would be the black king's free space in general, i.e. its free space computed by taking the rook's and the white king's position into account, like the attribute *Black King's Available Squares*.

In figure 3.31, the attribute has a value of 12 in the first example because the black king has a free space of 3 squares per 4 squares. In the second example, the value is 2, and it does not matter if the rook can be taken or not. In the third example, as the rook and the black king are on the same file, the biggest side (i.e. the left one) is chosen for the calculation, which yields 24 free squares.

3.9.6 BK's Available Squares

How many squares are available to the black king for the next move?

This attribute counts the number of fields where the black king is free to go in the next move. If its value is 0, then the game is over because the black king has



$n = 0, draw$ $n = 2$ $n = 5$

Figure 3.32: Examples for the Attribute "BK's Available Squares"

the move but has no field where he can go. It is then a draw or a lose for the black, like on the first board of figure 3.32. Otherwise and in general, the less free squares the black king has available, the better it is for the winning side. There may though be an exception, as a value of zero may mean a draw, which is a bad result for the white side. The value of the attribute is computed like in the following: From the maximal value of eight free squares around the black king, we substract the number of squares which are either not on the board (if the black king is on an edge or in a corner), or too near too the white king (if the black king is only 2 squares away from the white king), or threatened by the rook (i.e. on the same file or the same rank as the rook). All the remaining squares are free squares for the black king.

Chapter 4

Knowledge Pertinence

Once we have defined the attributes, grouped them into 6 sets and calculated their values for all the positions of the endgame, we have to look at the 6 new formed databases to see how useful they are.

4.1 Purpose

The goal of the new database is to represent the same informations of the original database in a different way.
To achieve that, we use the higher level knowledge defined in the last chapter. Instead of storing the position of the pieces on the board with the corresponding distance to win, we store the values of chosen attributes, which need less storage place, hoping that we will be able to recover from them the original distance to win. Our hope is to find a combination of attributes that gives us a new database as small as possible that can then be compressed as good as possible, in which the depth of win is as well-defined as in the original one.

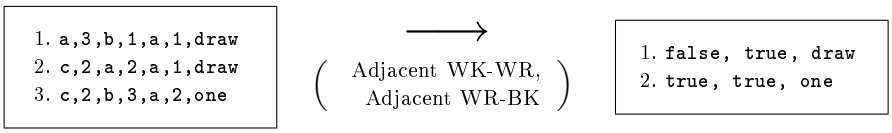
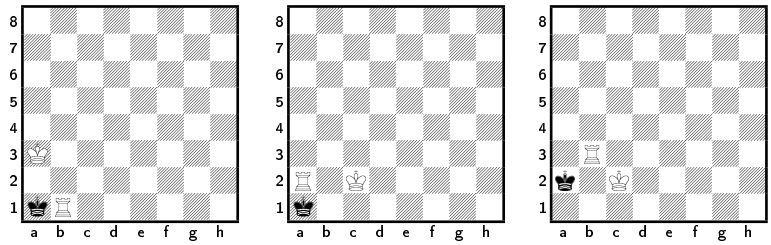


Figure 4.1: Attributes Transformation Reduction Example

The transformation of the position data in attribute values brings a reduction of the number of lines in the database, because several positions can be described by the same attributes values. An example is shown on figure 4.1 and explained on figure 4.2: We consider a simple set with only the 2 attributes *Adjacent WK*

- *WR* and *Adjacent WR - BK*. The position of the figures on the first board is then represented by $(false, true)$, as well as on the second board, and they both are drawn. The position on the third board is represented by $(true, true)$ and is mate in one move. If we consider only these 3 positions, we can conclude that all positions with the values $(false, true)$ are drawn, and all positions with the values $(true, true)$ are mate in one. In other words, such a 3-line primary database would be transformed with the attribute set (*Adjacent WK - WR*, *Adjacent WR - BK*) to a new database containing only two lines. Therefore we have reduced the number of lines from 3 to 2 while transforming the database.



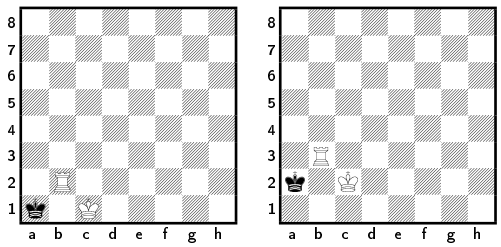
Adjacent WK-WR:	`false`	`false`	`true`
Adjacent WR-BK:	`true`	`true`	`true`
Distance to win:	`draw`	`draw`	`one`
Attributes values:	(`false`,`true`,`draw`)	(`false`,`true`,`draw`)	(`true`, `true`,`one`)

Figure 4.2: Attributes Transformation Reduction Example in details

Since we do not want to lose information from the database, we cannot let ambiguous combinations appear. Ambiguous positions, as explained in section 1.4, are positions for which the same attribute values correspond to different distances to win. We illustrate that in figure 4.3 with the same attribute set as in the previous example. As you can see, the two shown positions have the same attribute values but a different depth of win. Thus the combination (`true`,`true`) cannot be classified as "draw" or as "mate in one" without misclassifying a position, therefore we say that the positions are ambiguous for the used attribute set. Thus we have to store in a special file all the positions which can lead to ambiguous combinations. This file should be as small as possible, because it cannot be compressed in the same way as the new database. So it represents a gap in our compression process, unless it is empty or almost empty, what is the case when the attributes distinguish the depth of win efficiently. Nevertheless, it is interesting to look at the structure of this error file, to see which depths of win raise problems for a given attribute combination. If the file is empty (perfect case), the attributes are well chosen, then we have not lost information by changing the database and all

the positions have an image in the new database.

In this chapter, we will look at the database generated of each attribute set, and compare the properties of each of them. For the ones which are interesting, we will then look at the repartition of the errors on the depth of win, to see which depths of win present a problem.



Adjacent WK-WR:	`true`	`true`
Adjacent WR-BK:	`true`	`true`
Distance to win:	`draw`	`one`
Attributes values:	(`true`,`true`,`draw`)	(`true`, `true`,`one`)

Figure 4.3: Ambiguous Positions Example

4.2 General Statistics

Six different databases were created from the initial database. To know the accuracy of the attribute sets used to generate them, we look first at general properties., which we describe in the following: First the size of the database, measured as the number of lines in the new database; this amount is smaller than or equal to the number of lines in the primary database, because the attribute transformation reduces the number of lines, as explained in the previous section. We measure the size of the database with the number of lines it contains, because the size of a line is not important: This is indeed about proportional to the number of attributes, and some attributes may be left aside by the next step of the compression, which relies on knowledge based analysis. Second the number of errors, which tells how many positions from the initial database are transformed in a combination of attributes values with an ambiguous depth of win (as explained in the previous section, particularly on figure 4.3); the smaller this number is, the better the set describes the different positions. Third the correctness, which is the proportion of positions in the primary database which will not be ambiguous after the transformation; this number is almost the inverse of the number of

errors. Fourth the compression, which tells how many times smaller is the new database; this number takes into account the number of errors; since the errors are not compressed, the compression can just come from the transformed part of the database. Please keep in mind that the compression here does not directly imply anything on the disc space which will be needed to store the new database, since we only consider the number of lines of each file, and not the size of each line. These properties for all databases are summarised in the table 4.1.

Attr. set	DB size	Errors	Correctness	Compression
Bain	1864	23 585	15.94 %	0.91
Morales	3	26 736	4.70 %	0.95
Sadikov	5 567	17 442	37.83 %	0.82
Torres	0	28 056	0 %	1
My Selection	22 531	2 272	91.90 %	0.88
All Attributes	28 056	0	100 %	1

Table 4.1: Properties of the six new databases

The results shown are various. For the first set, i.e. the attributes from [BMS95], the new database is small, what could be good if it would be caused by a high compression, but the correctness is quite low. So there are a lot of ambiguous positions, but despite that the compression is not totally bad. We will have a look on the repartition of the errors in the next section. For the second set, [Mor97], only three combinations are unambiguous! That is very few, and they are in fact only for draw positions. That may be because the system which discovered the attributes was trained with a simple non-optimal strategy. And the small number of attributes makes it also complicated. The third set, [Sad06], is much better. The correctness is still smaller than 50 %, but the compression is the best one. Here also, it is interesting to analyse the repartition of the errors. The fourth set, with the attributes from the chess machine, gives very bad results. No position at all can be clearly recognised. This result is surely because the machine does not take care of the exact position of the figures, but just tests if their distances respect some criteria. Moreover the machine cannot play from any start position, and our adaptation of the attributes so that every position can be computed may introduce ambiguities. The fifth set, with special attributes selected to distinguish the small depth of win, is quite efficient. The number of errors is small, what leads to a big database. The compression is smaller than the one of the set of Sadikov, but since the errors are fewer, we will be able to compress better in the next step. The last set, which regroups all the attributes, goes further in the same

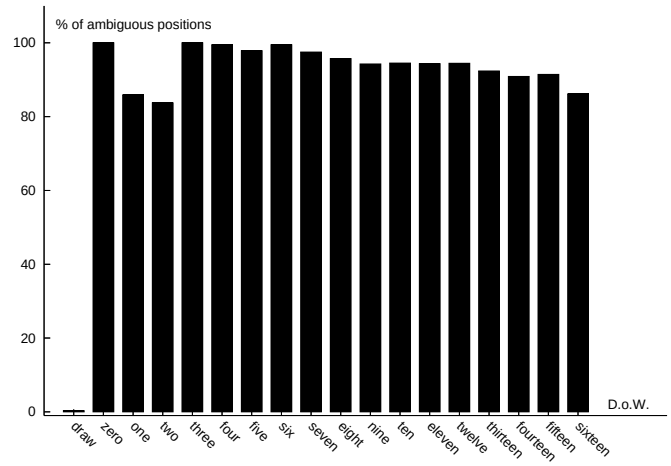


Figure 4.4: Part of ambiguous positions for each depth of win for Bain's attributes

direction: There is no ambiguous position (zero error) but also no compression: Each position of the initial database is represented by one particular combination.

In the next section, we will analyse the error repartition for the databases for which it may be interesting, i.e. for Bain, Sadikov, and my selection. Since all the attributes together do not generate any error, there is nothing to analyse there.

4.3 Differentiation Power of the Attributes Combinations

The graphs shown in this section are built that way: In the building process of each database, if a combination of attributes values happens to appear more than one time for different depth of win, all the positions that pointed to this combination are marked as ambiguous and stored in the error file. Then we counted simply the number of position for each depth of win, and the division by the number of positions for this distance to win in the original database gives us the proportions that are shown on the graphs.

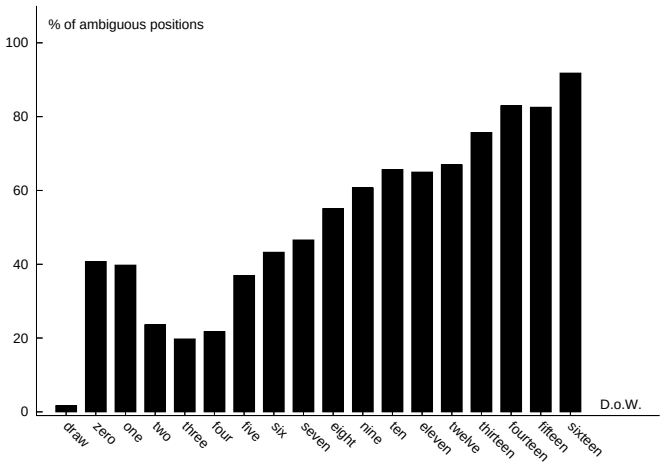


Figure 4.5: Part of ambiguous positions for each depth of win for Sadikov's attributes

4.3.1 Bain

As we can see on the figure 4.4, the attributes from Bain can perfectly distinguish the draw positions: No error appears. But for the other positions, the attributes are quite bad: more than 80% of the not draw positions are ambiguous. Some depth of win are even totally unclear (100 % ambiguous). We can conclude from this result that the attributes of this set are only able to identify surely positions facts (like illegal positions, mate positions, pat positions, rook takeable ...) but cannot distinguish well more subtile patterns which could give informations about the depth of win, if it is not draw. With such a proportion of errors, this set is not interesting for proceeding further analysis with WEKA.

4.3.2 Sadikov

Sadikov's attributes compose the best set from the four littereture sources. As seen in figure 4.5, the draw positions are almost all distinct after the transformation. For larger distances to win, the part of univocal positions decreases slowly, what is logic, since it is more difficult to take two positions of a large distance to win apart. Until a distance to win of seven, the proportion stays under 50%, which is good. But then the error proportions become larger and since the amount of

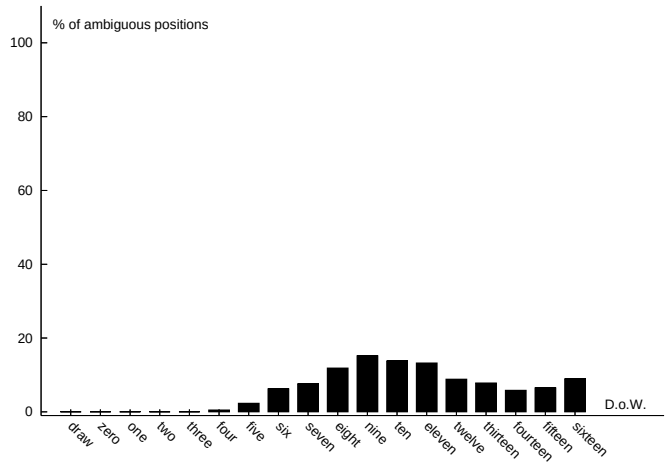


Figure 4.6: Part of ambiguous positions for each depth of win for the attributes of My Selection

positions for high depths of win is also large, the whole correctness of the database is low (37.83%). Still, it is a good set of attributes, because the compression is the best one.

4.3.3 My Attributes

The attributes I selected were chosen to detect at best the lower distances to win. I proceeded this way: starting from a small amount of attributes selected intuitively, I added some other by looking at the positions that the current selection could not detect. The results in figure 4.6 are really good: For every depth of win, less than 20% of the positions are ambiguous! And as expected, the lower distances to win (draw to three) are clearly set aside, and four and five almost perfectly detected.

Chapter 5

Knowledge-Based Compression

Now that we have generated three useable databases with higher level knowledge, we build a decision tree of each of them with variable properties to obtain a database representation as small as possible. After the analyses of last chapter, we only consider the new databases from *Sadikov*, *My Selection*, and *All Attributes*.

This last step of the compression relies on knowledge-based algorithms. Such algorithms allow us to find the most important attributes in the database, i.e. the ones which can best differentiate the depth of win. Thus the size of the compressed database will not depend directly on the number of attributes it contains, but rather on the quality of its attributes. The number of lines of the database has on the contrary much impact on the final size because it represents the amount of instances which have to be classified. With a lot of instances, the knowledge algorithms have less chances to find a simple representation of the database, i.e. in our case, the decision trees which we build would probably be more complex.

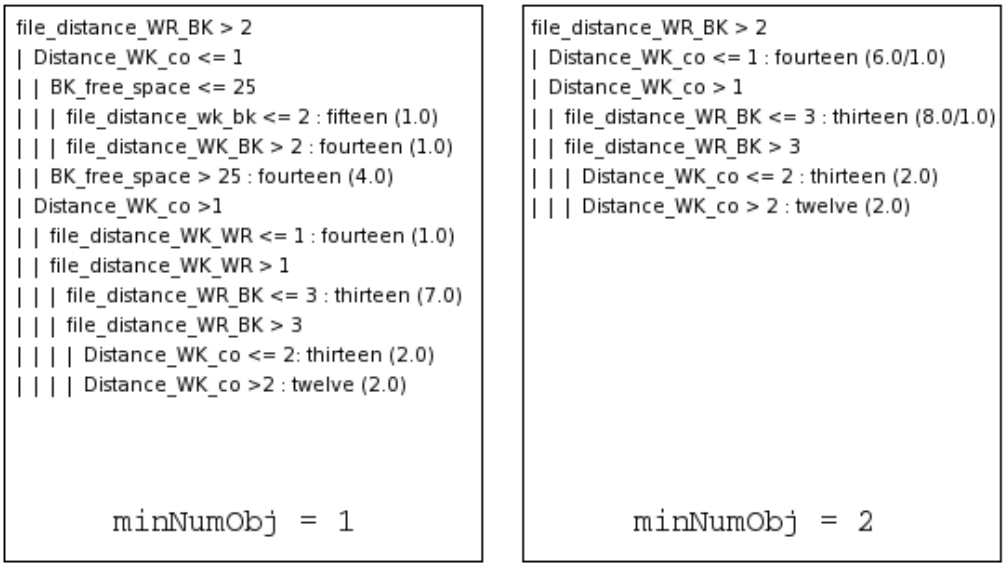
5.1 Perfect Tree

5.1.1 Preliminaries

First we want to build a perfect tree from each database, to compare generally the complexity of the different databases. To do that, we use the software WEKA[WF05]. This software implements amongst other algorithms a c4.5 decision tree classifier[1], which allows us to build a decision tree with adapted options.

[1] `weka.classifiers.trees.J48`

This classifier accepts two main options: We can set the minimum number of instance per leaf (short `minNumObj`), and if the tree should be pruned or not. We chose to build an unpruned tree with a `minNumObj` of 1, to obtain a classification as perfect as possible. The pruning of a decision tree reduces its complexity and the time needed to build it, of course at the price of some information loss. The `minNumObj` defines the level of detail of the tree.



Beneath the main node file_distance_WR_BK > 2 *there are 18 instances: 2×* `twelve`*, 9×* `thirteen`*, 6×* `fourteen` *and 1×* `fifteen`*. Since there are different depths of win with more than 2 instances, the algorithm has to look further for another division. Beneath the second node* distance_WK_co ≤ 1*, there are 6 instances: 5×* `fourteen` *and 1×* `fifteen`*. For a* `minNumObj` *of 1, the algorithm has to look further to distinguish* `fourteen` *from* `fifteen`*. But for a* `minNumObj` *of 2, the one instance with* `fifteen` *is negligible; in that case, the algorithm does not search further, it classifies the 6 instances as* `fourteen`*, specifies that one error has occurred and continues then with the 12 other instances. The same procedure is applied for the rest of the tree: each time an instance is alone on a leaf, it is neglected if the* `minNumObj` *is 2.*

Figure 5.1: Decision tree Example for 2 values of `minNumObj`

In figure 5.1, you can see on the left side a part of a c4.5 unpruned decision tree with a `minNumObj` of 1. On the right side, the corresponding part of the tree with a `minNumObj` of 2 is shown. You can notice that each time that a leaf contains only one object on the left, this part of the tree is cut off on the right.

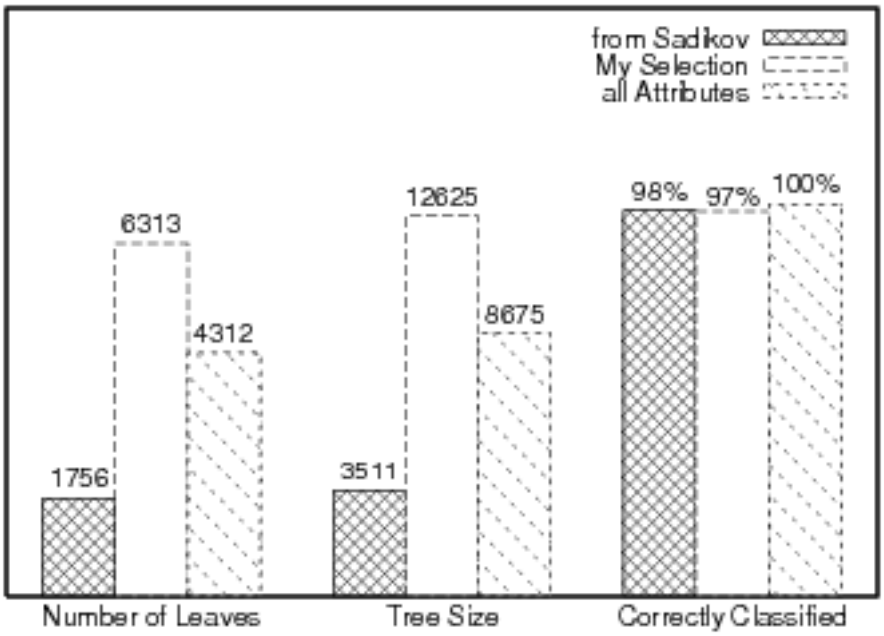


Figure 5.2: Properties of the Decision Trees

5.1.2 Perfect Tree Properties of the New Databases

With the options we chose (unpruned and a `minNumObj` of 1), WEKA should generate perfect trees. The general properties of the different generated trees can be seen in figure 5.2.

As you can see, the trees are not perfect, but almost. Less than 3% of the instances are incorrectly classified. To still be able to recover precise information from the database, we have to save the errors separately. This can be achieved with another tool from WEKA, the weka unsupervised filter to remove misclassified instances[2]. This is a filter that extracts or removes instances which are incorrectly classified by a specified classifier, the J48 Decision Tree in our case. Thus we store the incorrectly classified instances in a special file, and then the decision tree does not contain any error. The properties of the perfect trees show also that the database from Sadikov gives a much simpler tree than the other (smaller tree and less leaves). This is what we expected, since the Sadikov's database is much smaller than the two others. Another interesting result is that the database containing all attributes gives a tree which is less complex than the one of the attributes from my selection, although the database from all attributes is slightly bigger. This

[2] `weka.filters.unsupervised.instances.RemoveMisclassified`

can be explained by the fact that the algorithm from WEKA can select better attributes amongst all attributes to build the tree than amongst my selection, because more attributes are available, so that the small size difference can be compensated. But the scale of this phenomena is too small to bring the tree complexity of all attributes to the level of the one of Sadikov.

Except for Sadikov, you can see that the trees are quite complex. For this reason we tried to build decision trees with a larger `minNumObj` (and also more errors), to see if the space gain due to the tree being less complex brings more than the loss due to the error file being larger.

5.2 Variation of the Tree Complexity

We tried different values for the *Least Instance Number* per leaf, from 1 to 128. We could have gone further if the graph would have shown no clear tendance. The evolution of the complexity of the trees is shown in the graph 5.3. You can find in Appendix A the detail of the output returned by WEKA while computing the decision trees.

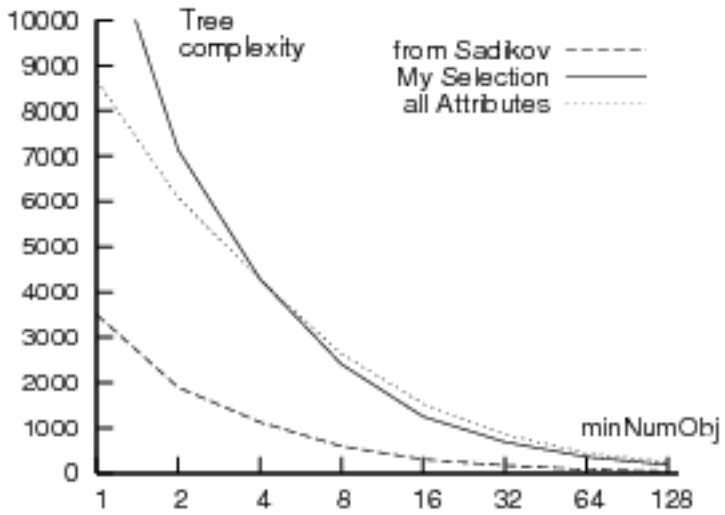


Figure 5.3: Complexity of the Decision Trees depending on the `minNumObj`

The complexity of the tree from Sadikov's attributes set is always simpler than the other two. But the tree from my selection becomes as simple as the one from all attributes when `minNumObj` grows. The reason may be the difficulty to build an (almost) perfect tree with less attributes, implying a much larger complexity at

the beginning, which then decreases quickly. To complete this graph, we also have to look at the accuracy of the tree. The evolution of the proportion of correctly classified instances is shown in figure 5.4.

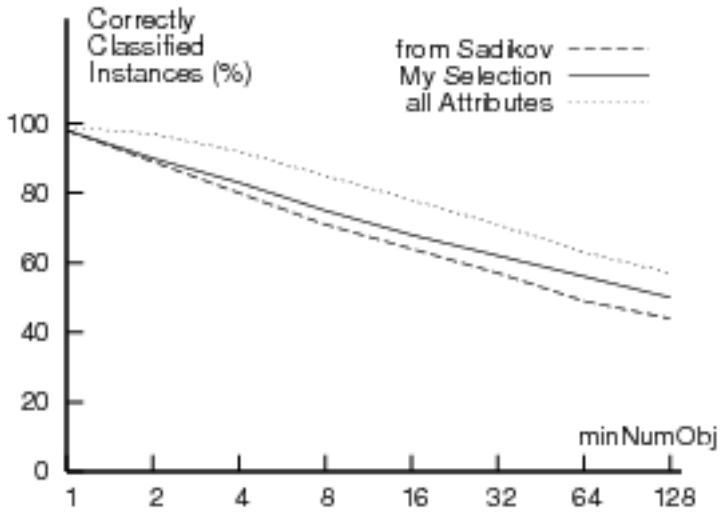


Figure 5.4: Accuracy of the Decision Trees depending on the `minNumObj`

As seen in the figures, the classification errors grow linearly while the complexity of the trees decreases exponentially. There are slightly less errors made by the set "my selection" in comparison to the set from Sadikov's paper, but the behaviour of the three lines is the same in general. The analysis of the results shown on the two graphs lets us expect that a particular value of the `minNumObj` should give a good combination of a simplified decision tree with few classification errors. The complexity of the trees reflects the size needed to store them, but does not take into account the space needed to save the errors. Depending on the size of the misclassified instances, it is better to to keep a complex tree with very few errors or to simplify the tree by letting instances be misclassified. To know exactly in which case we are, we have to analyse the space needed to store the whole new databases.

5.3 Memory Space Needed by the New Databases

What we are interested in is to to load the whole database in the computer memory, to be able to use it at high speed. The important measure of the quality of our compression is thus the disc space required to store the new databases.

Unlike the initial database, the new databases do not consist in only one file. They are composed of three files:

1. The position error file, which contains the ambiguous positions and was obtained while calculating the attribute values;
2. The classification error file, which contains the combinations of attribute values misclassified by the decision tree, obtained while building the decision tree with WEKA.
3. The tree file, which is the most important part of the database, which consists of a text file containing a representation of a the tree as it was shown in the figure 5.1.

This representation is not space-optimised because lots of characters are used to build a human understable design, but these character repetitions are compressed very efficiently by the compression algorithm so that this representation is really effective. The first file does not depend on the properties of the tree, since the ambiguous positions are already calculated and cannot be influenced by the `minNumObj` of the tree. On the contrary, the size of the two other files directly depend on the tree properties. The second file should have its size getting larger as the complexity of the tree decreases, since more errors are made. The third file should on the contrary become smaller, as less place is needed to store a simple tree. Like in the previous section, we let vary the `minNumObj` because it is a simple way of controlling the complexity of the tree and also the number of errors made by the classifier. The results are shown on figure 5.5.

The size of the database is measured as follows: The three files are grouped together in an archive and compressed with the algorithm bzip2. bzip2 uses the Burrows-Wheeler transform to convert frequently recurring character sequences into strings of identical letters. This method allows us to store the decision tree directly as a graphical output from WEKA in a text file, since the character repetitions used to draw the tree can be efficiently compressed, so that they do not have much influence on the compressed file size. We used the Unix tar utility[3]. The size in bytes of the bzipped tar archive is used to measure the size of the database.

As you can see, the influence of the `minNumObj` parameter depends on the attributes set. This result could be forseen, because the errors are written as attribute value combinations, therefore the number of attributes directly influences

[3]GNU tar version 1.15.1, with the -bzip2 option, see `http://www.gnu.org/software/tar/`

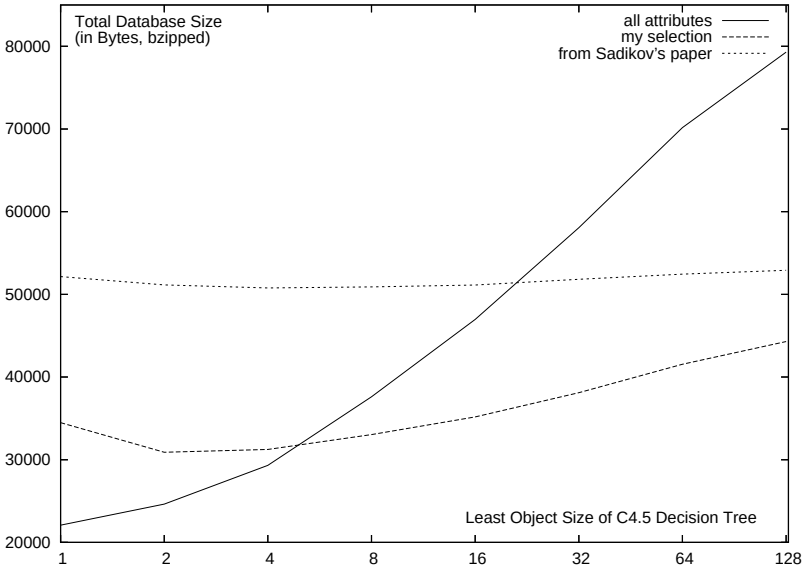


Figure 5.5: Database sizes for different `minNumObj`

the disc space needed to save each error made while building the tree (lots of attributes mean more values to store i.e. more disc space needed). We could save the chess positions corresponding to the misclassified combinations, but it would not be necessarily smaller, as one combination may represent more than one position, what implies that the expense to retrieve the corresponding positions is not worth it.

As we said, the first file does not depend on the number of instances per leaf, therefore the number of position errors just displace the curves upwards or downwards on the graph. Since the set from Sadikov produced the most position errors, it is not surprising that the corresponding curve is the upper one at the beginning. It then stays about at the same level: That is due to the small complexity of the tree, which cannot get much simpler, so there is no big change. The smallest size is obtained for a `minNumObj` of 4. For the two other on the contrary the evolution is quite interesting: The set *My Selection* has a minimum for a `minNumObj` of two, and then goes slowly upwards, so the best condition to store the database with this set is with a tree with two instances per leaf. The set *All Attributes* on the contrary always grows, because of the big space needed to store the errors. But it starts at the lowest level. The smallest possible size of the new database is thus

for a `minNumObj` of one by the set *All Attributes*, and consists of 22087 Bytes. We will discuss this value in the next section.

5.4 Compression Results

The results obtained in the previous section should be compared to the size of the other representations of the database to know their quality. This comparison is shown in figure 5.6. For each database, we proceeded identically to measure the size: we grouped first the files (or just one for the primary database) together in a tar-archive, and then compressed it with bzip.

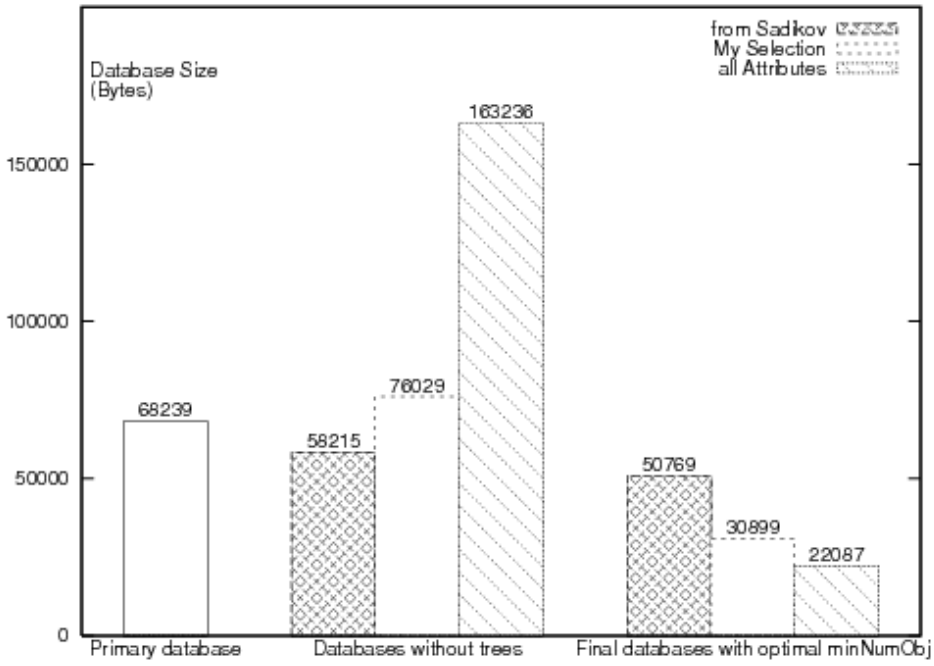


Figure 5.6: Size of the different Databases

As you can see, the primary database was 68239 Bytes big, and our best result (for all attributes and a `minNumObj` of 1) reduces the database to 22087 Bytes. So the best compression we achieved with our method is 32%, which means a compression rate of 3,09. Moreover, you can observe the efficiency of the knowledge-based compression for the set *All Attributes*: From the biggest database after the attributes transformation (due to the high number of attributes, which means a lot of values to store), it becomes the smallest at the end. In contrary, the database of the set *Sadikov* stays about the same; that should be caused by the small size

of the database and the big size of the position error file, letting few possibilities for the decision tree to compress it.

The compression we achieved could be surely enhanced with some methods adapted to decision trees. For example, some incorrectly classified combinations could be stored at low space cost if the position in the tree would be stored instead of the whole combination for each error (in case of several errors in the same area of the tree). Another possibility would be in some case to store the position of the three figures on the board for the misclassified combinations instead of storing directly the combinations of the attributes values. To do that, we would have to undo the attributes transformation, i.e. to retrieve the chess positions corresponding to these combinations. Depending on the number of positions for each combination, the disc space needed could become smaller, because a chess position has only six values, compared to up to 33 for the set *All Attributes*. But that was not the main goal of our work, it can just help to improve the efficiency while implementing the methods. We showed that the knowledge-based compression could help a lot to reduce the size of the database, but that it is in a great part depending on the quality of the attributes.

Chapter 6

Conclusion

Chess endgame databases allow to play perfectly all endgames with less than 6 figures, but their large size is a big handicap which cannot be solved with standard compression method. Knowledge based methods from the Artificial Intelligence domain are promising for this task. In this report, we detailed our efforts to obtain a compressed form of the 3 figures KRK endgame database relying on chess attributes. Basing our research on interesting works using KRK patterns, we have developed a knowledge consisting in 33 attributes. Thus we were able to transform the primary database in a bunch of new ones by using different attribute sets. Thanks to Data Mining decision trees, we could then save the new databases very optimally, obtaining in the best case a compression 3 times better than standard compression methods.

This result shows that it is surely possible to use Artificial Intelligence method to store and then to use chess endgames databases. Saved under this new form, the tablebases can be easier loaded in the main memory of computers and with an appropriate algorithm efficiently used to play perfectly chess endgames. The compression rate we achieved may be improved by adding other attributes, with which the database would be better described, although the patterns we used are known by chess grand masters as the most relevant ones for the KRK endgame. However it may be more efficient to rather split the database in subsets and to specify different attributes sets for each subset, since the attributes have different weight depending on the state of the endgame (mate in 2 moves or rather mate in 15 moves for example). The errors may also be stored more optimally, either with the help of appropriate attributes or, for the database lines misclassified in the tree, by specifying the branch of the decision tree where the errors happened.

The same compression system should also work for bigger chess endgame

databases. It requires however that the database had been deeply studied, so that efficient attributes are available. Standard attributes like figures distances, board distances etc... are easy to define, but important figures relations on key positions on the board are specific for each endgame. Since our method rely on the quality of the attributes, the results which are to be espected depend also on the previous studies about the endgame in question and on the quality of the patterns they found. An efficient system which can detect which characteristics are the most important to measure the depth of win and deliver best quality attributes would enable the use of our method for any other endgame. Unfortunately, such a system does not exist yet, but some endgames have already been deeply studied. For example, the KQKR endgame database could surely be efficiently compressed with our method.

Some other general methods, which do not rely on specific chess patterns and use nonetheless knowledge based compression, may be a good idea to avoid the issue of finding adapted attributes. Such systems already exist for big and complex databases (like SPARTAN, [BGR01]), and some similar processes, maybe adapted to chess endgames, should bring good compression possibilities for larger chess tablebases.

Appendix A

Decision Trees Output

In this appendix, we show the whole output returned by WEKA while computing the decision trees. For each of the three databases *Sadikov*, *My Selection* and *All Atributes*, the properties of the trees as well as the confusion matrix are included for the `minNumObj` parameter taking the values 1, 2, 4, 8, 16, 32, 64 and 128.

Sadikov

One Object per Leaf

```
Options: -U -M 1

J48 unpruned tree
------------------

Number of Leaves  :  1756

Size of the tree :  3511


Time taken to build model: 11 seconds
Time taken to test model on training data: 0.73 seconds

=== Error on training (and test) data ===

Correctly Classified Instances        5478               98.4013 %
Incorrectly Classified Instances        89                1.5987 %
Kappa statistic                          0.9824
Mean absolute error                      0.0025
Root mean squared error                  0.0355
Relative absolute error                  2.49   %
Root relative squared error             15.7808 %
Total Number of Instances             5567
```

```
=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 1077    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    a = draw
    0   16    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    0    0   32    5    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    c = one
    0    0    0  137    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    d = two
    0    0    0    1   59    1    0    0    0    0    0    0    0    0    0    0    0    0 |    e = three
    0    0    0    0    1  140    0    0    0    0    0    0    0    0    0    0    0    0 |    f = four
    0    0    0    0    0    0  247    3    5    0    0    0    0    0    0    0    0    0 |    g = five
    0    0    0    0    0    1    0  280    6    0    2    0    0    0    0    0    0    0 |    h = six
    0    0    0    0    0    0    2    0  308    2    0    0    0    0    0    0    0    0 |    i = seven
    0    0    0    0    0    0    0    0    0  500    2    2    3    0    0    0    0    0 |    j = eight
    0    0    0    0    0    0    1    0    0    2  418    4    0    0    0    0    0    0 |    k = nine
    0    0    0    0    0    0    3    0    0    1    0  383    3    3    0    0    0    0 |    l = ten
    0    0    2    0    0    1    0    0    0    0    0    1  468    2    0    0    0    0 |    m = eleven
    0    0    0    0    0    0    0    0    0    0    2    0    4  476    3    0    0    0 |    n = twelve
    0    0    0    0    0    0    0    0    0    0    0    0    1    2  391    0    1    0 |    o = thirteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0   11  331    0    0 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    3  190    1 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1   25 |    r = sixteen
```

Two Objects per Leaf

```
Options: -U -M 2

J48 unpruned tree
------------------

Number of Leaves  :  950

Size of the tree :  1899


Time taken to build model: 8.15 seconds
Time taken to test model on training data: 0.86 seconds

=== Error on training (and test) data ===

Correctly Classified Instances        4929               88.5396 %
Incorrectly Classified Instances       638               11.4604 %
Kappa statistic                          0.8739
Mean absolute error                      0.0178
Root mean squared error                  0.0943
Relative absolute error                 17.6199 %
Root relative squared error             41.9787 %
Total Number of Instances             5567


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 1076    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0 |    a = draw
    0   13    0    3    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
```

```
   1    0   28    6    0    0    0    1    1    0    0    0    0    0    0    0    0    0 |    c = one
   1    0    1  134    1    0    0    0    0    0    0    0    0    0    0    0    0    0 |    d = two
   0    0    0    3   54    2    1    1    0    0    0    0    0    0    0    0    0    0 |    e = three
   2    0    0    0    2  128    8    1    0    0    0    0    0    0    0    0    0    0 |    f = four
   1    0    0    0    4    6  222    9   10    2    0    1    0    0    0    0    0    0 |    g = five
   2    0    1    0    1    3    7  248   12    7    5    1    1    1    0    0    0    0 |    h = six
   2    0    0    1    0    1    5   11  270   16    2    1    1    1    1    0    0    0 |    i = seven
   2    0    0    0    0    0    4    8    8  451   15   13    5    0    0    1    0    0 |    j = eight
   1    0    0    0    0    1    1    4    6   17  372   11    8    2    2    0    0    0 |    k = nine
   0    0    1    0    0    0    2    6    7   20   15  317   14   10    0    0    1    0 |    l = ten
   1    0    4    0    0    2    2    0    0    9   11   15  412   14    4    0    0    0 |    m = eleven
   1    0    0    0    0    3    1    0    2    3    8    8   20  418   16    4    1    0 |    n = twelve
   1    0    0    0    0    0    0    1    0    1    2   10    5   39  320   14    2    0 |    o = thirteen
   1    0    0    0    0    0    0    0    0    1    0    4    3   12   29  279   13    0 |    p = fourteen
   0    0    0    0    0    0    0    0    0    0    0    0    0    1    6   17  170    1 |    q = fifteen
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    8   17 |    r = sixteen
```

Four Objects per Leaf

```
Options: -U -M 4

J48 unpruned tree
------------------

Number of Leaves  :  567

Size of the tree :  1133


Time taken to build model: 7.19 seconds
Time taken to test model on training data: 1.29 seconds

=== Error on training (and test) data ===

Correctly Classified Instances        4434               79.6479 %
Incorrectly Classified Instances      1133               20.3521 %
Kappa statistic                          0.7761
Mean absolute error                      0.0311
Root mean squared error                  0.1247
Relative absolute error                 30.7596 %
Root relative squared error             55.4648 %
Total Number of Instances             5567


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 1071    0    0    0    0    0    0    2    0    0    0    0    2    0    0    0    2    0 |    a = draw
    0   11    0    5    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    1    0   28    6    0    0    0    1    1    0    0    0    0    0    0    0    0    0 |    c = one
    3    0    1  129    4    0    0    0    0    0    0    0    0    0    0    0    0    0 |    d = two
    1    0    0    3   51    3    2    0    1    0    0    0    0    0    0    0    0    0 |    e = three
    2    0    0    1    2  119   13    1    1    0    2    0    0    0    0    0    0    0 |    f = four
    1    0    0    1    3   13  202   14   14    3    2    1    1    0    0    0    0    0 |    g = five
    2    0    1    3    2   12   15  200   28   12    9    2    1    1    1    0    0    0 |    h = six
```

```
   4    0    0    2    0    2   19   19  225   24    7    5    2    2    1    0    0    0 |    i = seven
   3    0    0    0    0    0    6   16   31  385   35   18   11    1    1    0    0    0 |    j = eight
   2    0    0    0    0    1    3   10    9   37  320   21   11    9    1    0    1    0 |    k = nine
   3    0    0    0    0    0    4    7   10   24   30  254   32   25    2    2    0    0 |    l = ten
   3    0    4    0    0    2    2    2    2   14   21   26  363   29    3    3    0    0 |    m = eleven
   1    0    0    0    0    3    2    0    2    5   15   14   42  348   38   11    4    0 |    n = twelve
   5    0    0    0    0    0    1    2    0    1    4    9   16   41  291   23    2    0 |    o = thirteen
   0    0    0    0    0    0    0    0    0    0    0    4    8    7   33  264   26    0 |    p = fourteen
   0    0    0    0    0    0    0    0    0    0    0    0    0    1    6   28  158    2 |    q = fifteen
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    0   10   15 |    r = sixteen
```

Eight Objects per Leaf

```
Options: -U -M 8

J48 unpruned tree
------------------

Number of Leaves  :  302

Size of the tree :  603


Time taken to build model: 6.27 seconds
Time taken to test model on training data: 0.72 seconds

=== Error on training (and test) data ===

Correctly Classified Instances        3974               71.3849 %
Incorrectly Classified Instances      1593               28.6151 %
Kappa statistic                          0.6851
Mean absolute error                      0.0427
Root mean squared error                  0.1462
Relative absolute error                 42.2893 %
Root relative squared error             65.0342 %
Total Number of Instances             5567


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 1066    0    0    6    0    0    0    0    1    0    0    0    0    2    0    0    2    0 |    a = draw
    0   13    0    3    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    0    0   20   11    0    0    0    4    1    0    0    0    0    0    1    0    0    0 |    c = one
    1    3    0  121   10    0    2    0    0    0    0    0    0    0    0    0    0    0 |    d = two
    2    0    0    0   51    4    3    0    1    0    0    0    0    0    0    0    0    0 |    e = three
    2    0    0    0    9  107   14    1    1    0    2    0    5    0    0    0    0    0 |    f = four
    2    0    2    0   16   20  156   16   23   13    3    0    3    0    1    0    0    0 |    g = five
    6    0    0    0    4   14   20  188   29   17    3    3    3    1    1    0    0    0 |    h = six
    6    0    0    1    4    3   14   33  184   41   18    2    1    1    4    0    0    0 |    i = seven
    3    0    0    0    1    0   10   17   34  356   40   16   21    8    1    0    0    0 |    j = eight
    2    0    0    0    0    5    3   14   13   45  282   26   15   18    1    0    1    0 |    k = nine
    3    0    1    0    3    0    1   11   17   44   62  171   43   27    8    2    0    0 |    l = ten
    5    0    5    0    0    1    0    3    3   21   45   17  303   51   18    2    0    0 |    m = eleven
    0    0    1    0    0    3    0    1    3    7   33   20   34  306   65   10    2    0 |    n = twelve
```

```
    5    0    0    0    0    0    0    3    0    3    9    7   17   45  267   36    3    0 |    o = thirteen
    0    0    0    0    0    0    0    0    0    0    1    2   13    7   50  240   29    0 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    1    3   52  135    4 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    1   16    8 |    r = sixteen
```

Sixteen Objects per Leaf

```
Options: -U -M 16

J48 unpruned tree
------------------

Number of Leaves  :  156

Size of the tree :  311


Time taken to build model: 3.92 seconds
Time taken to test model on training data: 0.59 seconds

=== Error on training (and test) data ===

Correctly Classified Instances        3563               64.0022 %
Incorrectly Classified Instances      2004               35.9978 %
Kappa statistic                          0.6031
Mean absolute error                      0.052
Root mean squared error                  0.1613
Relative absolute error                 51.5116 %
Root relative squared error             71.776  %
Total Number of Instances             5567


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 1076    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0 |    a = draw
    0    9    0    7    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    3    0   25    3    0    0    0    4    1    0    1    0    0    0    0    0    0    0 |    c = one
    6    8    7  110    3    0    2    1    0    0    0    0    0    0    0    0    0    0 |    d = two
    4    0    0   12   17    3   12    8    5    0    0    0    0    0    0    0    0    0 |    e = three
    4    0    0    1    5   76   33    9    7    0    0    0    5    1    0    0    0    0 |    f = four
    5    0    2    2    6   13  156   31   15   11    5    4    4    1    0    0    0    0 |    g = five
    8    0    1    2    1   12   29  151   48   22    1    2    9    3    0    0    0    0 |    h = six
    8    0    0    1    3    4   20   41  142   61   15    9    4    2    2    0    0    0 |    i = seven
    5    0    1    0    0    0   13   21   30  305   36   34   41   14    7    0    0    0 |    j = eight
    5    0    3    0    4    0   12    7   22   70  196   30   36   26   14    0    0    0 |    k = nine
    3    0    1    0    0    0    4    7   24   31   66  136   59   40   15    7    0    0 |    l = ten
    4    0    5    0    0    0    0    0    5   19   26   22  292   64   27   10    0    0 |    m = eleven
    3    0    1    0    0    0    0    3    7    7   23   12   46  271   74   36    2    0 |    n = twelve
    3    0    0    0    0    0    0    0    1    1    6    8   26   41  234   73    2    0 |    o = thirteen
    3    0    0    0    0    0    0    0    0    0    0    1   16   10   48  241   23    0 |    p = fourteen
    3    0    0    0    0    0    0    0    0    0    0    0    1    3    9   59  113    7 |    q = fifteen
    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    2   10   13 |    r = sixteen
```

Thirty two Objects per Leaf

```
Options: -U -M 32

J48 unpruned tree
------------------

Number of Leaves  :  89

Size of the tree :  177


Time taken to build model: 3.87 seconds
Time taken to test model on training data: 0.59 seconds

=== Error on training (and test) data ===

Correctly Classified Instances        3176               57.0505 %
Incorrectly Classified Instances      2391               42.9495 %
Kappa statistic                          0.5261
Mean absolute error                      0.0602
Root mean squared error                  0.1735
Relative absolute error                 59.5527 %
Root relative squared error             77.1752 %
Total Number of Instances             5567


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 1062    0    0    6    0    0    0    0    0    0    0    0    0    0    0    9    0    0 |    a = draw
    0    0    0   16    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    1    0    0   25    0    0    0    1    0    3    0    0    7    0    0    0    0    0 |    c = one
    1    0    0  132    0    0    3    0    0    1    0    0    0    0    0    0    0    0 |    d = two
    3    0    0   16    0    3   31    0    0    8    0    0    0    0    0    0    0    0 |    e = three
    4    0    0    4    0   61   36   25    4    1    1    0    5    0    0    0    0    0 |    f = four
    7    0    0    8    0   13  132   36   15   32    0    0    9    1    2    0    0    0 |    g = five
   10    0    0    6    0    2   27  140   28   58    5    0   10    1    2    0    0    0 |    h = six
   11    0    0    6    0    0   48   48   85   66   23    1   18    3    2    1    0    0 |    i = seven
    6    0    0    0    0    0    8   27   33  261   65   17   49   26   15    0    0    0 |    j = eight
    7    0    0    0    0    0   25   10    9   68  198    8   68   22    9    1    0    0 |    k = nine
    6    0    0    0    0    0    5    3   16   45   83   77   96   42   16    0    4    0 |    l = ten
    8    0    0    0    0    0    0    0    3   22   52   18  278   46   38    9    0    0 |    m = eleven
    4    0    0    0    0    0    0    0    8    9   42   27   77  178   91   45    4    0 |    n = twelve
    2    0    0    0    0    0    0    0    1    0   12    6   26   50  227   66    5    0 |    o = thirteen
    0    0    0    0    0    0    0    0    0    0    2    2    7    8   66  222   35    0 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    1    0    0    2   12   57  123    0 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    3   23    0 |    r = sixteen
```

Sixty four Objects per Leaf

```
Options: -U -M 64

J48 unpruned tree
------------------
```

```
Number of Leaves  :  45

Size of the tree :  89


Time taken to build model: 3.06 seconds
Time taken to test model on training data: 1.46 seconds

=== Error on training (and test) data ===

Correctly Classified Instances        2744               49.2905 %
Incorrectly Classified Instances      2823               50.7095 %
Kappa statistic                          0.4406
Mean absolute error                      0.0684
Root mean squared error                  0.185
Relative absolute error                 67.7427 %
Root relative squared error             82.3111 %
Total Number of Instances             5567


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 1071    0    0    6    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    a = draw
    0    0    0   16    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    1    0    0    6    0   25    0    1    3    0    0    0    1    0    0    0    0    0 |    c = one
    1    0    0  104   22   10    0    0    0    0    0    0    0    0    0    0    0    0 |    d = two
    3    0    0    6   23   19    5    0    5    0    0    0    0    0    0    0    0    0 |    e = three
    4    0    0    4    8   56   31   26    6    5    0    0    0    0    1    0    0    0 |    f = four
    7    0    0    8   22   41   62   59   33   16    0    1    3    1    2    0    0    0 |    g = five
   10    0    0    6    5    4   12  138   44   55    2    1    7    1    4    0    0    0 |    h = six
   12    0    0    6   20    3    6   56   88   80   10   14    9    4    4    0    0    0 |    i = seven
    6    0    0    0    0    1    2   34   53  243   51   33   43   19   22    0    0    0 |    j = eight
    8    0    0    0    0   12    3   12   45   85  129   22   76   16   17    0    0    0 |    k = nine
    6    0    0    0    0    1    5    3   20   82   65   49   62   63   26   11    0    0 |    l = ten
    9    0    0    0    0    5   12    1   26   37   97   30  146   37   72    2    0    0 |    m = eleven
    8    0    0    0    0    1   10    0    6   34   30   14   62  168   92   60    0    0 |    n = twelve
    7    0    0    0    0    0    1    0    1   15   23    9   23   50  187   79    0    0 |    o = thirteen
   11    0    0    0    0    0    0    0    0    1   15    1    5   12   63  198   36    0 |    p = fourteen
    8    0    0    0    0    0    0    0    0    0    1    0    0   21    6   77   82    0 |    q = fifteen
    1    0    0    0    0    0    0    0    0    0    0    0    0    1    0    4   20    0 |    r = sixteen
```

Hundread twenty eight Objects per Leaf

```
Options: -U -M 128

J48 unpruned tree
------------------

Number of Leaves  :  23

Size of the tree :  45


Time taken to build model: 2.09 seconds
```

```
Time taken to test model on training data: 0.67 seconds

=== Error on training (and test) data ===

Correctly Classified Instances        2476               44.4764 %
Incorrectly Classified Instances      3091               55.5236 %
Kappa statistic                          0.3867
Mean absolute error                      0.073
Root mean squared error                  0.191
Relative absolute error                 72.2328 %
Root relative squared error             84.9952 %
Total Number of Instances             5567


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 1071    0    0    6    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    a = draw
    0    0    0   16    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    1    0    0    6    0   19    6    0    0    1    3    0    0    0    1    0    0    0 |    c = one
    1    0    0  104    0   31    1    0    0    0    0    0    0    0    0    0    0    0 |    d = two
    3    0    0    6    0   29   18    0    0    0    0    0    5    0    0    0    0    0 |    e = three
    4    0    0    4    0   45   45   30    0    6    5    0    1    0    1    0    0    0 |    f = four
    7    0    0    8    0   45   46   71    0   37   16    0   21    0    3    1    0    0 |    g = five
   10    0    0    6    0    7   24  101    0  108   19    0    8    0    5    1    0    0 |    h = six
   12    0    0    6    0   23    0   42    0  127   61    1   32    0    7    0    1    0 |    i = seven
    6    0    0    0    0    0   27   17    0  257  121    4   45    4   24    1    1    0 |    j = eight
    8    0    0    0    0    5    9   16    0   96  165   44   30   22   25    2    3    0 |    k = nine
    6    0    0    0    0    0    6    0    0   93   80   67   49   39   30   17    6    0 |    l = ten
    9    0    0    0    0    0   29    2    0   48  118   45   89   62   63    5    4    0 |    m = eleven
    8    0    0    0    0    0   11    0    0   30   96   44   37   86  103   47   23    0 |    n = twelve
    7    0    0    0    0    0    1    0    0   15   39    3   29   14  196   58   33    0 |    o = thirteen
   11    0    0    0    0    0    0    0    0    1   19    0    8    0  100  128   75    0 |    p = fourteen
    8    0    0    0    0    0    0    0    0    0    1    0    0    0   18   47  121    0 |    q = fifteen
    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    3   22    0 |    r = sixteen
```

My Selection

One Object per Leaf

```
Options: -U -M 1

J48 unpruned tree
------------------

Number of Leaves  :  6313

Size of the tree :  12625


Time taken to build model: 18.31 seconds
Time taken to test model on training data: 0.47 seconds

=== Error on training (and test) data ===
```

```
Correctly Classified Instances       22100               98.0871 %
Incorrectly Classified Instances       431                1.9129 %
Kappa statistic                          0.9787
Mean absolute error                      0.0027
Root mean squared error                  0.0368
Relative absolute error                  2.719  %
Root relative squared error             16.4899 %
Total Number of Instances            22531


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 1927    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    a = draw
    0   27    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    0    0   68   10    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    c = one
    0    0    0  246    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    d = two
    0    0    0    0   81    0    0    0    0    0    0    0    0    0    0    0    0    0 |    e = three
    0    0    0    0    4  180    6    5    0    0    0    0    0    0    0    0    0    0 |    f = four
    0    0    0    0    0    6  441    4    0    0    0    0    0    0    0    0    0    0 |    g = five
    0    0    0    0    0    0   21  521    1    5    0    0    0    0    0    0    0    0 |    h = six
    0    0    1    0    0    0    6    6  578    3    9    0    2    0    0    0    0    0 |    i = seven
    0    0    0    0    0    0    0   22    0 1177    7    5    3    0    0    0    0    0 |    j = eight
    0    0    0    0    0    0    0    0    9    6 1331    1    3    1    0    0    0    0 |    k = nine
    0    0    0    0    0    0    0    0    0    2   19 1577    7    2    2    0    0    0 |    l = ten
    0    0    0    0    0    0    0    0    3    3   18   24 2184   12    6    0    0    0 |    m = eleven
    0    0    0    0    0    0    0    0    0    3    2   14   19 2868   15    2    0    0 |    n = twelve
    0    0    0    0    0    0    0    0    0    0    0    6    8   14 3317   22    2    1 |    o = thirteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    4   31 3611   16    0 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0   11   12 1672    1 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    4  294 |    r = sixteen
```

Two Objects per Leaf

```
Options: -U -M 2

J48 unpruned tree
------------------

Number of Leaves  :  3567

Size of the tree :  7133


Time taken to build model: 14.32 seconds
Time taken to test model on training data: 0.44 seconds

=== Error on training (and test) data ===

Correctly Classified Instances       20365               90.3866 %
Incorrectly Classified Instances      2166                9.6134 %
Kappa statistic                          0.8929
Mean absolute error                      0.015
Root mean squared error                  0.0867
Relative absolute error                 15.0691 %
```

```
Root relative squared error            38.8198 %
Total Number of Instances           22531


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 1927    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    a = draw
    0   27    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    0    0   77    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0 |    c = one
    0    0    8  234    1    2    0    0    1    0    0    0    0    0    0    0    0    0 |    d = two
    0    0    0    3   73    3    1    0    1    0    0    0    0    0    0    0    0    0 |    e = three
    0    0    0    1    2  173   10    7    1    1    0    0    0    0    0    0    0    0 |    f = four
    0    0    0    1    4    9  414   14    5    0    1    1    1    1    0    0    0    0 |    g = five
    0    0    0    0    2    5   29  476   20   10    2    2    1    0    0    1    0    0 |    h = six
    0    0    1    0    0    0    8   19  528   17   20    4    7    1    0    0    0    0 |    i = seven
    0    0    0    0    0    0   11   35   31 1059   44   17   10    6    1    0    0    0 |    j = eight
    0    0    0    0    0    0    4    6   11   45 1212   28   21   17    6    1    0    0 |    k = nine
    0    0    0    0    0    0    2    0    4   23   49 1432   43   27   22    7    0    0 |    l = ten
    0    0    0    0    0    0    3    2    9   38   41   81 1923   86   47   19    1    0 |    m = eleven
    0    0    0    0    0    0    1    1    2   13   14   47   89 2588  120   42    6    0 |    n = twelve
    0    0    0    0    0    0    1    1    1    2    2   23   43  111 3003  155   25    3 |    o = thirteen
    0    0    0    0    0    0    0    0    0    1    0    7   19   47  118 3383   79    8 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    1    0    3    7   30   84 1559   12 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    4   17  277 |    r = sixteen
```

Four Objects per Leaf

```
Options: -U -M 4

J48 unpruned tree
------------------

Number of Leaves  :  2148

Size of the tree :  4295


Time taken to build model: 10.03 seconds
Time taken to test model on training data: 0.43 seconds

=== Error on training (and test) data ===

Correctly Classified Instances       18609               82.5929 %
Incorrectly Classified Instances      3922               17.4071 %
Kappa statistic                          0.8062
Mean absolute error                      0.0269
Root mean squared error                  0.116
Relative absolute error                 26.9983 %
Root relative squared error             51.961  %
Total Number of Instances            22531


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
```

```
 1927    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    a = draw
    2   25    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    0    0   76    1    1    0    0    0    0    0    0    0    0    0    0    0    0    0 |    c = one
    1    0    9  226    3    4    1    1    1    0    0    0    0    0    0    0    0    0 |    d = two
    0    0    0    1   61    8    4    2    5    0    0    0    0    0    0    0    0    0 |    e = three
    0    0    0    0    5  166   14    7    3    0    0    0    0    0    0    0    0    0 |    f = four
    0    0    0    0    2   17  378   22   13    5    2    3    7    2    0    0    0    0 |    g = five
    0    0    1    0    2    8   42  426   22   28    3    5    7    3    0    1    0    0 |    h = six
    1    0    4    0    0    4   23   36  436   47   24   11   11    8    0    0    0    0 |    i = seven
    0    0    0    0    0    0   10   30   55  932   90   41   30   22    4    0    0    0 |    j = eight
    0    0    0    0    0    0    4    7   31   69 1076   67   54   33    8    1    1    0 |    k = nine
    0    0    0    0    0    0    5    7   20   42   60 1234  109   70   51   11    0    0 |    l = ten
    0    0    0    0    0    0    4    7   12   40   61  116 1749  148   81   32    0    0 |    m = eleven
    0    0    0    0    0    0    5    1    4   16   37   73  169 2346  185   80    7    0 |    n = twelve
    0    0    0    0    0    0    1    0    6    5    9   40   92  234 2692  247   36    8 |    o = thirteen
    0    0    0    0    0    0    1    0    2    1    2   19   19   81  215 3144  169    9 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    0    2    7   22   28  146 1466   25 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    8   40  249 |    r = sixteen
```

Eight Objects per Leaf

```
Options: -U -M 8

J48 unpruned tree
------------------

Number of Leaves  :  1207

Size of the tree :  2413


Time taken to build model: 6.72 seconds
Time taken to test model on training data: 0.41 seconds

=== Error on training (and test) data ===

Correctly Classified Instances       16885               74.9412 %
Incorrectly Classified Instances      5646               25.0588 %
Kappa statistic                          0.7207
Mean absolute error                      0.0381
Root mean squared error                  0.1381
Relative absolute error                 38.2275 %
Root relative squared error             61.8297 %
Total Number of Instances            22531


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 1927    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    a = draw
    2   25    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    1    0   72    2    0    0    0    0    3    0    0    0    0    0    0    0    0    0 |    c = one
    0    5    8  220    8    2    2    0    1    0    0    0    0    0    0    0    0    0 |    d = two
    3    0    0    6   58    6    4    0    2    2    0    0    0    0    0    0    0    0 |    e = three
    0    0    0    0   15  140   25   10    1    0    4    0    0    0    0    0    0    0 |    f = four
```

```
    1    0    1    0    9   17  339   33   17    5    6   11    5    7    0    0    0    0 |    g = five
    0    0    0    5    7   11   58  362   46   30    8    9    5    4    0    3    0    0 |    h = six
    0    0    4    1    4    7   28   33  371   65   36   22   17   14    1    2    0    0 |    i = seven
    0    0    0    0    1    1    9   40   79  833  114   68   30   31    7    1    0    0 |    j = eight
    0    0    0    0    1    0    3    6   44  134  911  112   56   67   12    4    1    0 |    k = nine
    0    0    0    0    0    0    8   10   27   70   85 1036  177  104   65   24    3    0 |    l = ten
    0    0    0    0    0    0    6    3   12   57   98  150 1527  224  105   63    5    0 |    m = eleven
    0    0    0    0    0    0    0    3    1   31   44   87  235 2049  322  131   20    0 |    n = twelve
    0    0    0    0    0    0    0    0    2    7   22   63  120  237 2442  426   47    4 |    o = thirteen
    0    0    0    0    0    0    0    0    0    2    3   14   18  105  277 2996  238    9 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    0    1    3   16   40  238 1350   48 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    3   11   57  227 |    r = sixteen
```

Sixteen Objects per Leaf

```
Options: -U -M 16

J48 unpruned tree
------------------

Number of Leaves  :  628

Size of the tree :  1255


Time taken to build model: 4.5 seconds
Time taken to test model on training data: 0.39 seconds

=== Error on training (and test) data ===

Correctly Classified Instances       15374               68.2349 %
Incorrectly Classified Instances      7157               31.7651 %
Kappa statistic                          0.6459
Mean absolute error                      0.0476
Root mean squared error                  0.1543
Relative absolute error                 47.7554 %
Root relative squared error             69.1067 %
Total Number of Instances            22531


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 1918    0    0    6    0    3    0    0    0    0    0    0    0    0    0    0    0    0 |    a = draw
    0   10    5   12    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    0    0   61   13    0    1    0    0    3    0    0    0    0    0    0    0    0    0 |    c = one
    0   10    5  212   15    2    1    0    1    0    0    0    0    0    0    0    0    0 |    d = two
    0    0    0    2   43   17   12    5    2    0    0    0    0    0    0    0    0    0 |    e = three
    0    0    0    1   11  137   26   10    6    0    4    0    0    0    0    0    0    0 |    f = four
    0    0    3    2    5   24  316   39   18   17    5    9    5    8    0    0    0    0 |    g = five
    0    0    2    5    3   14   63  333   53   40    8   13    4    4    4    2    0    0 |    h = six
    0    0    4    4    3    1   42   55  295   96   46   19   16   15    5    4    0    0 |    i = seven
    0    0    1    4    3    0   12   65   68  733  144   66   60   30   25    3    0    0 |    j = eight
    0    0    0    3    5    0    5    8   31  156  797  109  114   90   23    9    1    0 |    k = nine
    0    0    0    3    3    1   20   14   24   93  155  825  233  103  103   31    1    0 |    l = ten
```

```
    0    0    0    0    5    1    5   12   12   54   99  131 1453  259  147   66    6    0 |    m = eleven
    0    0    0    0    0    0    5   12    3   36   44  127  322 1801  374  163   36    0 |    n = twelve
    0    0    0    0    0    0    3    4    2   14   15   68  129  327 2262  476   65    5 |    o = thirteen
    0    0    0    0    0    0    0    3    0    3    0   33   35  112  398 2784  272   22 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    0    1    2   16   50  360 1189   78 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    3    2   88  205 |    r = sixteen
```

Thirty two Objects per Leaf

```
Options: -U -M 32

J48 unpruned tree
------------------

Number of Leaves  :  348

Size of the tree :  695


Time taken to build model: 3.36 seconds
Time taken to test model on training data: 0.37 seconds

=== Error on training (and test) data ===

Correctly Classified Instances       13964               61.9768 %
Incorrectly Classified Instances      8567               38.0232 %
Kappa statistic                          0.5759
Mean absolute error                      0.0555
Root mean squared error                  0.1665
Relative absolute error                 55.5963 %
Root relative squared error             74.5645 %
Total Number of Instances            22531


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 1918    0    3    0    0    0    3    0    3    0    0    0    0    0    0    0    0    0 |    a = draw
    0   10    7   10    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    0    0   56    8    0    0    1    0   10    0    3    0    0    0    0    0    0    0 |    c = one
    0    9   19  203    0    2    9    1    3    0    0    0    0    0    0    0    0    0 |    d = two
    0    0    5   11   17   20   26    0    1    0    0    1    0    0    0    0    0    0 |    e = three
    0    0    4    9    0   89   58   22    0    4    0    5    0    0    4    0    0    0 |    f = four
    0    0    7   34    6   23  238   80    4   35    4    8    7    3    2    0    0    0 |    g = five
    0    0   14    5    7   14   55  326   54   44   10    3   10    3    2    1    0    0 |    h = six
    0    1    5    0    3    5   60   58  268   65   37   38   21   32   12    0    0    0 |    i = seven
    0    1    4    0    0    2   27   54  117  604  179   90   79   31   24    2    0    0 |    j = eight
    0    4    3    1    0    0   13   25   41  180  734  100  124   76   46    4    0    0 |    k = nine
    0    3    3    0    0    1   26   29    5  112  173  703  251  154  110   36    3    0 |    l = ten
    0    3    0    0    0    1    9   32   10   56  134  159 1290  254  208   84   10    0 |    m = eleven
    0    8    0    0    0    0    3   23    8   36   68  129  357 1480  498  263   50    0 |    n = twelve
    0    1    0    1    0    0    4    5    3    9   23   77  131  305 2108  597  102    4 |    o = thirteen
    0    0    0    0    0    0    1    3    0    2    2   29   28   86  479 2648  350   34 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    0    1    1    3   49  455 1120   67 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    4   29  113  152 |    r = sixteen
```

Sixty four Objects per Leaf

```
Options: -U -M 64

J48 unpruned tree
------------------

Number of Leaves  :  185

Size of the tree :  369


Time taken to build model: 2.66 seconds
Time taken to test model on training data: 0.33 seconds

=== Error on training (and test) data ===

Correctly Classified Instances       12551               55.7055 %
Incorrectly Classified Instances      9980               44.2945 %
Kappa statistic                          0.5057
Mean absolute error                      0.0626
Root mean squared error                  0.1769
Relative absolute error                 62.7248 %
Root relative squared error             79.2007 %
Total Number of Instances            22531


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 1918    0    0    6    0    0    3    0    0    0    0    0    0    0    0    0    0    0 |    a = draw
    0    0    0   22    0    0    0    5    0    0    0    0    0    0    0    0    0    0 |    b = zero
    0    0   24   25    0    0    1   25    0    3    0    0    0    0    0    0    0    0 |    c = one
    0    0    9  221    0    0    9    7    0    0    0    0    0    0    0    0    0    0 |    d = two
    0    0    5   19    0    2   23    5   17    4    0    1    5    0    0    0    0    0 |    e = three
    0    0    4   13    0   26   34   74    4    5    0    5   30    0    0    0    0    0 |    f = four
    0    0    3   67    0   21  142  116   46   30    3    9   14    0    0    0    0    0 |    g = five
    0    0    6   11    0   24   85  195  109   67    9   16   15    5    6    0    0    0 |    h = six
    0    0    3   16    0    2   54   25  223  134   36   41   50   10   10    1    0    0 |    i = seven
    0    0    4   33    0    0   50   24  108  579  167  120   65   32   28    4    0    0 |    j = eight
    0    0    3   20    0    0   27   21   37  204  616  111  179   78   47    8    0    0 |    k = nine
    0    0    3   12    0    0   16   38    4  136  165  550  335  177  133   35    5    0 |    l = ten
    0    0    0    6    0    0   15   25    6  120   98  214 1101  343  215   99    8    0 |    m = eleven
    0    0    0    8    0    0    2   32    4   67   72  124  342 1336  586  306   44    0 |    n = twelve
    0    0    0    2    0    0    3    7    0    6   35   78  157  251 2017  683  125    6 |    o = thirteen
    0    0    0    0    0    0    1   12    0    0   14   19   54   88  528 2482  435   29 |    p = fourteen
    0    0    0    0    0    0    0    9    0    0    6    2    3    1   36  569  982   88 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1   13  145  139 |    r = sixteen
```

Hundread twenty eight Objects per Leaf

```
Options: -U -M 128

J48 unpruned tree
```

```
------------------

Number of Leaves  :  95

Size of the tree :  189


Time taken to build model: 2.35 seconds
Time taken to test model on training data: 0.31 seconds

=== Error on training (and test) data ===

Correctly Classified Instances       11362               50.4283 %
Incorrectly Classified Instances     11169               49.5717 %
Kappa statistic                          0.4465
Mean absolute error                      0.0681
Root mean squared error                  0.1846
Relative absolute error                 68.2962 %
Root relative squared error             82.6432 %
Total Number of Instances            22531


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 1918    0    0    6    0    0    0    0    0    0    0    0    0    0    0    3    0    0 |    a = draw
    0    0    0   27    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    0    0    0   74    0    0    0    0    0    0    0    0    3    0    0    1    0    0 |    c = one
    0    0    0  207    0    0   36    0    2    0    0    0    0    0    0    1    0    0 |    d = two
    0    0    0   24    0    0   28    0   20    3    0    0    1    0    0    5    0    0 |    e = three
    0    0    0   19    0    0  129    0   17   10    0    0   12    0    0    8    0    0 |    f = four
    0    0    0   36    0    0  281    0   49   39    7    8   13    6    0   12    0    0 |    g = five
    0    0    0   49    0    0  141    0   74  136   37   53   18   23    3   14    0    0 |    h = six
    0    0    0   52    0    0   73    0  192   70   83   52   56    7    7   13    0    0 |    i = seven
    0    0    0   50    0    0  100    0  130  319  231  158  146   39   25   16    0    0 |    j = eight
    0    0    0   29    0    0   37    0   70   99  567  126  220  134   45   24    0    0 |    k = nine
    0    0    0   27    0    0   44    0   80   69  206  414  363  127  125  143   11    0 |    l = ten
    0    0    0   17    0    0   44    0   48   49  236  222  884  424  183  125   18    0 |    m = eleven
    0    0    0   15    0    0   26    0    7   53  126  129  340 1353  463  356   55    0 |    n = twelve
    0    0    0    9    0    0    5    0    2   17   20  114  245  335 1709  747  162    5 |    o = thirteen
    0    0    0    1    0    0    2    0    0    1    0   30   66  208  427 2440  464   23 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    0    1    7   28   20  599  963   78 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    2   10  171  115 |    r = sixteen
```

All Attributes

One Object per Leaf

```
Options: -U -M 1

J48 unpruned tree
------------------

Number of Leaves  :  4372
```

```
Size of the tree :  8675


Time taken to build model: 132.17 seconds
Time taken to test model on training data: 3.26 seconds

=== Error on training (and test) data ===

Correctly Classified Instances       28044               99.9572 %
Incorrectly Classified Instances        12                0.0428 %
Kappa statistic                          0.9995
Mean absolute error                      0.0001
Root mean squared error                  0.0057
Relative absolute error                  0.0661 %
Root relative squared error              2.5702 %
Total Number of Instances            28056


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 2796    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    a = draw
    0   27    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    0    0   78    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    c = one
    0    0    0  246    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    d = two
    0    0    0    0   81    0    0    0    0    0    0    0    0    0    0    0    0    0 |    e = three
    0    0    0    0    0  198    0    0    0    0    0    0    0    0    0    0    0    0 |    f = four
    0    0    0    0    0    0  471    0    0    0    0    0    0    0    0    0    0    0 |    g = five
    0    0    0    0    0    0    0  592    0    0    0    0    0    0    0    0    0    0 |    h = six
    0    0    0    0    0    0    0    0  683    0    0    0    0    0    0    0    0    0 |    i = seven
    0    0    0    0    0    0    0    0    0 1433    0    0    0    0    0    0    0    0 |    j = eight
    0    0    0    0    0    0    0    0    0    0 1712    0    0    0    0    0    0    0 |    k = nine
    0    0    0    0    0    0    0    0    0    0    0 1985    0    0    0    0    0    0 |    l = ten
    0    0    0    0    0    0    0    0    0    0    0    0 2854    0    0    0    0    0 |    m = eleven
    0    0    0    0    0    0    0    0    0    0    0    0    6 3589    2    0    0    0 |    n = twelve
    0    0    0    0    0    0    0    0    0    0    0    0    0    0 4194    0    0    0 |    o = thirteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    2 4551    0    0 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    2 2164    0 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0  390 |    r = sixteen
```

Two Objects per Leaf

```
Options: -U -M 2

J48 unpruned tree
------------------

Number of Leaves  :  3060

Size of the tree :  6078


Time taken to build model: 119.06 seconds
Time taken to test model on training data: 3.66 seconds
```

```
=== Error on training (and test) data ===

Correctly Classified Instances       27113               96.6389 %
Incorrectly Classified Instances       943                3.3611 %
Kappa statistic                          0.9625
Mean absolute error                      0.0056
Root mean squared error                  0.0527
Relative absolute error                  5.5845 %
Root relative squared error             23.632  %
Total Number of Instances            28056


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 2796    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    a = draw
    1   25    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    0    0   78    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    c = one
    0    0    0  246    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    d = two
    0    0    0    2   77    1    1    0    0    0    0    0    0    0    0    0    0    0 |    e = three
    0    0    0    0    2  193    3    0    0    0    0    0    0    0    0    0    0    0 |    f = four
    0    0    1    0    3    3  460    1    2    1    0    0    0    0    0    0    0    0 |    g = five
    0    0    0    0    3    3   13  545   16    5    3    1    2    1    0    0    0    0 |    h = six
    0    0    2    1    1    0    4   12  637   14    7    4    1    0    0    0    0    0 |    i = seven
    0    0    0    0    0    0    3    7   19 1373   21    9    1    0    0    0    0    0 |    j = eight
    0    0    0    0    0    0    1    3    6   30 1635   22   12    2    1    0    0    0 |    k = nine
    0    0    0    0    0    0    2    0    4   14   38 1885   32    8    2    0    0    0 |    l = ten
    0    0    0    0    0    0    2    0    4    5   14   34 2717   57   16    4    1    0 |    m = eleven
    0    0    0    0    0    0    1    0    2    1    4   13   46 3466   54    8    2    0 |    n = twelve
    0    0    0    0    0    0    0    0    1    1    1    6   18   62 4038   66    1    0 |    o = thirteen
    0    0    0    0    0    0    0    0    0    0    0    1    1    8   61 4453   29    0 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    1    4   45 2110    6 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1   10  379 |    r = sixteen
```

Four Objects per Leaf

```
Options: -U -M 4

J48 unpruned tree
------------------

Number of Leaves  :  2161

Size of the tree :  4289


Time taken to build model: 107.8 seconds
Time taken to test model on training data: 3.94 seconds

=== Error on training (and test) data ===

Correctly Classified Instances       25818               92.0231 %
Incorrectly Classified Instances      2238                7.9769 %
Kappa statistic                          0.9109
Mean absolute error                      0.0129
Root mean squared error                  0.0802
```

```
Relative absolute error                12.9297 %
Root relative squared error            35.9585 %
Total Number of Instances           28056


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 2796    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    a = draw
    1   25    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    0    0   77    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0 |    c = one
    1    0    0  241    2    2    0    0    0    0    0    0    0    0    0    0    0    0 |    d = two
    0    0    0    2   69    1    4    2    3    0    0    0    0    0    0    0    0    0 |    e = three
    0    0    0    1    1  181    7    7    1    0    0    0    0    0    0    0    0    0 |    f = four
    0    0    0    1    4   11  410   32   10    1    1    0    1    0    0    0    0    0 |    g = five
    0    0    0    1    2    5   28  502   25   18    5    2    4    0    0    0    0    0 |    h = six
    1    0    1    0    0    4    6   25  584   50    6    3    1    1    1    0    0    0 |    i = seven
    0    0    0    0    0    1    8   12   41 1291   53   18    6    2    1    0    0    0 |    j = eight
    0    0    0    0    0    0    2    3   12   66 1511   71   34    7    6    0    0    0 |    k = nine
    0    0    0    0    0    1    6    2   13   37   57 1752   81   27    8    1    0    0 |    l = ten
    0    0    0    0    0    1    2    0    8   11   24   81 2567  111   40    6    3    0 |    m = eleven
    0    0    0    0    0    0    2    2    1    3   13   27  114 3302  110   21    2    0 |    n = twelve
    0    0    0    0    0    2    2    0    3    2    5   20   25  149 3809  167   10    0 |    o = thirteen
    0    0    0    0    0    0    1    0    0    0    1    5    1   22  146 4294   83    0 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    2    5  101 2042   16 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0   25  365 |    r = sixteen
```

Eight Objects per Leaf

```
Options: -U -M 8

J48 unpruned tree
------------------

Number of Leaves  :  1329

Size of the tree :  2640


Time taken to build model: 108.24 seconds
Time taken to test model on training data: 3.32 seconds

=== Error on training (and test) data ===

Correctly Classified Instances       23903               85.1975 %
Incorrectly Classified Instances      4153               14.8025 %
Kappa statistic                          0.8347
Mean absolute error                      0.0233
Root mean squared error                  0.108
Relative absolute error                 23.4457 %
Root relative squared error             48.4216 %
Total Number of Instances            28056


=== Confusion Matrix ===
```

```
    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 2796    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    a = draw
    0   25    0    1    0    0    0    0    1    0    0    0    0    0    0    0    0    0 |    b = zero
    7    0   67    3    1    0    0    0    0    0    0    0    0    0    0    0    0    0 |    c = one
    0    0    0  239    2    2    0    0    3    0    0    0    0    0    0    0    0    0 |    d = two
    3    0    4    5   50    8    6    0    4    0    1    0    0    0    0    0    0    0 |    e = three
    0    2    1    0    0  165   19    6    2    0    0    0    0    3    0    0    0    0 |    f = four
    1    0    0    0   10   16  377   31   20    4    6    4    0    2    0    0    0    0 |    g = five
    0    1    0    2   18   20   27  439   34   21   20    4    1    1    3    1    0    0 |    h = six
    1    2    1    4    1    3   25   41  477   78   23   10    8    4    5    0    0    0 |    i = seven
    0    0    0    0    1    1    5   18   55 1160  122   46   15    6    4    0    0    0 |    j = eight
    0    0    0    1    0    0    0    6   17  112 1364  122   60   18   11    1    0    0 |    k = nine
    0    0    0    0    3    0    5   11   18   28  144 1525  174   62    9    5    1    0 |    l = ten
    0    2    1    0    0    0    5   10    5   15   67  152 2341  190   53   10    3    0 |    m = eleven
    0    3    0    0    0    0    3    0    6    1   23   71  204 3011  228   45    2    0 |    n = twelve
    0    0    0    0    0    0    0    0    7    3    9   17   66  281 3540  257   14    0 |    o = thirteen
    0    0    0    0    0    0    0    0    0    0    1    7   10   51  278 4057  147    2 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    0    0    1    6   10  198 1923   28 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    4   39  347 |    r = sixteen
```

Sixteen Objects per Leaf

```
Options: -U -M 16

J48 unpruned tree
------------------

Number of Leaves  :  775

Size of the tree :  1539


Time taken to build model: 76.46 seconds
Time taken to test model on training data: 2.87 seconds

=== Error on training (and test) data ===

Correctly Classified Instances       21924               78.1437 %
Incorrectly Classified Instances      6132               21.8563 %
Kappa statistic                          0.7558
Mean absolute error                      0.0337
Root mean squared error                  0.1298
Relative absolute error                 33.8614 %
Root relative squared error             58.1916 %
Total Number of Instances            28056


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 2793    0    0    0    0    3    0    0    0    0    0    0    0    0    0    0    0    0 |    a = draw
    0   10    0    1    5    0    5    0    1    0    0    5    0    0    0    0    0    0 |    b = zero
    6    0   65    0    6    1    0    0    0    0    0    0    0    0    0    0    0    0 |    c = one
    0    9    5  225    2    0    0    2    3    0    0    0    0    0    0    0    0    0 |    d = two
```

```
    5    0    4    5   35   16    4    5    3    4    0    0    0    0    0    0    0    0 |    e = three
    0    0    2    0    4  152   24   11    2    0    0    0    0    3    0    0    0    0 |    f = four
    0    0    0    3    1   24  343   47   25    9    6    1    4    0    2    6    0    0 |    g = five
    0    0    5    3    5   18   44  410   36   31   20    6    5    3    3    3    0    0 |    h = six
    1    0    3    5    0    8   32   49  374  126   23   33   15    6    8    0    0    0 |    i = seven
    0    0    6    1    0    0    5   13   77 1013  167   87   42   13    6    3    0    0 |    j = eight
    0    0    3    1    0    2    4    7   23  115 1209  181  108   37   21    1    0    0 |    k = nine
    0    0    2    0    0    2    9    4    6   41  168 1301  294  117   23   17    1    0 |    l = ten
    0    0    1    0    0    2    4    5    0    7   72  202 2099  342  109    5    6    0 |    m = eleven
    0    0    0    0    0    0    0    0    2    1   25   64  289 2720  418   76    2    0 |    n = twelve
    0    0    0    0    0    0    0    0    1    1    2   25  123  374 3286  353   28    1 |    o = thirteen
    0    0    0    0    0    0    0    0    0    0    0    3   10   65  446 3717  302   10 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    0    0    2    8   20  222 1881   33 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    8   90  291 |    r = sixteen
```

Thirty two Objects per Leaf

```
Options: -U -M 32

J48 unpruned tree
------------------

Number of Leaves  :  433

Size of the tree :  860


Time taken to build model: 57.79 seconds
Time taken to test model on training data: 2.97 seconds

=== Error on training (and test) data ===

Correctly Classified Instances       19903               70.9403 %
Incorrectly Classified Instances      8153               29.0597 %
Kappa statistic                          0.6753
Mean absolute error                      0.0439
Root mean squared error                  0.1481
Relative absolute error                 44.0794 %
Root relative squared error             66.3935 %
Total Number of Instances            28056


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 2787    0    0    6    0    0    3    0    0    0    0    0    0    0    0    0    0    0 |    a = draw
    0   10    6   11    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    0    0   56   12    0    0    7    0    0    0    0    0    0    3    0    0    0    0 |    c = one
    0    0   18  223    0    0    5    0    0    0    0    0    0    0    0    0    0    0 |    d = two
    0    6    0   22    0   13   30    1    5    4    0    0    0    0    0    0    0    0 |    e = three
    0    5    3    1    0  105   40   37    4    0    0    0    0    3    0    0    0    0 |    f = four
    0    8    3    5    0   20  252   80   52   28   10    0    4    3    0    6    0    0 |    g = five
    0    5    6    6    0    9   55  381   65   30   13    6    7    3    3    3    0    0 |    h = six
    0    0    8   11    0    7   23   56  340  153   31   20   11   17    6    0    0    0 |    i = seven
    0    0   10    3    0    0    2   59   50  913  248   91   29   15    9    4    0    0 |    j = eight
```

```
    0    0    7    0    0    0    2    7   50  176 1034  198  165   46   14   12    1    0 |    k = nine
    0    0    4    3    0    0    3    4   13   54  255 1136  312  155   31   14    1    0 |    l = ten
    0    0    1    0    0    0    5    7    0   21  150  292 1811  358  173   32    4    0 |    m = eleven
    0    0    0    0    0    0    0    0    5   11   38  109  366 2374  543  131   20    0 |    n = twelve
    0    0    0    0    0    0    0    0    0    4    4   31  106  464 3017  518   50    0 |    o = thirteen
    0    0    0    0    0    0    0    0    0    0    0   27    2   73  533 3517  381   20 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    1   49  377 1671   68 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    3  110  276 |    r = sixteen
```

Sixty four Objects per Leaf

```
Options: -U -M 64

J48 unpruned tree
------------------

Number of Leaves  :  227

Size of the tree :  452


Time taken to build model: 69.63 seconds
Time taken to test model on training data: 2.6 seconds

=== Error on training (and test) data ===

Correctly Classified Instances       17538               62.5107 %
Incorrectly Classified Instances     10518               37.4893 %
Kappa statistic                          0.5809
Mean absolute error                      0.054
Root mean squared error                  0.1643
Relative absolute error                 54.2515 %
Root relative squared error             73.6569 %
Total Number of Instances            28056


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 2787    0    0    0    0    0    3    4    0    2    0    0    0    0    0    0    0    0 |    a = draw
    0    0    5   22    0    0    0    0    0    0    0    0    0    0    0    0    0    0 |    b = zero
    0    0   25   40    0    0    1    0    0   12    0    0    0    0    0    0    0    0 |    c = one
    0    0    5  209    0    0   26    4    0    2    0    0    0    0    0    0    0    0 |    d = two
    0    0    0   23    0   13   35    0    2    4    3    0    1    0    0    0    0    0 |    e = three
    0    0    0   12    0  111   36   29    4    3    0    0    0    3    0    0    0    0 |    f = four
    0    0    0   77    0   39  180   57   25   39   23   10   15    0    0    6    0    0 |    g = five
    0    0    0   10    0   54  113  175   87   58   27   43   22    0    0    3    0    0 |    h = six
    0    0    0   16    0    3   42   55  287  145   80   15   19   14    6    1    0    0 |    i = seven
    0    0    0   16    0    6   34   74  111  784  297   51   38   10    6    6    0    0 |    j = eight
    0    0    1   15    0    0   14    5   39  281  952  116  150  113   16   10    0    0 |    k = nine
    0    0    7   11    0    0   13   55    3  189  370  588  367  274   60   48    0    0 |    l = ten
    0    0    3    7    0    0    6    5    0   80  242  200 1509  480  264   51    7    0 |    m = eleven
    0    0    9    3    0    0    0    0    0   29   73  111  399 2011  764  188   10    0 |    n = twelve
    0    0    7    3    0    0    0    0    0   12   13   39  108  406 2843  715   48    0 |    o = thirteen
    0    0    4    0    0    0    0    0    0    0    2    3   11   93  679 3255  469   37 |    p = fourteen
    0    0    0    0    0    0    0    0    0    0    0    1    0    4   64  429 1530  138 |    q = fifteen
```

```
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    4   10   84  292 |    r = sixteen
```

Hundread twenty eight Objects per Leaf

```
Options: -U -M 128

J48 unpruned tree
------------------

Number of Leaves  :  129

Size of the tree :  255


Time taken to build model: 51.57 seconds
Time taken to test model on training data: 2.97 seconds

=== Error on training (and test) data ===

Correctly Classified Instances       15872               56.5726 %
Incorrectly Classified Instances     12184               43.4274 %
Kappa statistic                          0.5134
Mean absolute error                      0.0606
Root mean squared error                  0.1741
Relative absolute error                 60.9312 %
Root relative squared error             78.0598 %
Total Number of Instances            28056


=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    q    r   <-- classified as
 2787    0    0    6    0    0    0    0    0    0    0    0    0    0    0    3    0    0 |    a = draw
    0    0    0   23    0    0    0    0    0    0    0    4    0    0    0    0    0    0 |    b = zero
    0    0    0   59    0    0   12    0    0    3    0    3    0    0    0    1    0    0 |    c = one
    0    0    0  188    0    0    3    0    0   19    0   35    0    0    0    1    0    0 |    d = two
    0    0    0    3    0   12   25    4    0   28    0    3    1    0    0    5    0    0 |    e = three
    0    0    0    6    0   66   39   57    6   13    0    0    0    3    0    8    0    0 |    f = four
    0    0    0   20    0   40  244   23   31   58    0   22   15    0    0   18    0    0 |    g = five
    0    0    0   19    0   24  139   87  102  113   10   52   32    0    0   14    0    0 |    h = six
    0    0    0   19    0    8   82   23  213  136   81   66   22   14    6   13    0    0 |    i = seven
    0    0    0   72    0    0  113   54  150  586  227  123   69   10   15   14    0    0 |    j = eight
    0    0    0   66    0    0   28   11   63  256  650  191  302   96   26   23    0    0 |    k = nine
    0    0    0   44    0    0    4   23   31   97  291  628  402  278   97   90    0    0 |    l = ten
    0    0    0   19    0    0    3    0   13   32  203  370 1468  430  228   83    5    0 |    m = eleven
    0    0    0   23    0    0    0    0    9   12   74  146  552 1734  819  213   15    0 |    n = twelve
    0    0    0   18    0    0    0    0    5    9   24   16  246  448 2565  821   42    0 |    o = thirteen
    0    0    0   15    0    0    0    0    0    0    4    0   88  187  661 3229  369    0 |    p = fourteen
    0    0    0    1    0    0    0    0    0    0    0    0    3   14   88  706 1269   85 |    q = fifteen
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    4   11  217  158 |    r = sixteen
```

Appendix B

Script Source Code

We implemented a script in Ruby[1] to read automatically the primary database and transform it practically in the new databases. This script allows us to chose a particular attribute set and to specify some other options, mainly for test purpose. The implementation is based on libraries which could be easily reused in other programms, if needed. The *Attribute Classes* contain the description of each attribute, and calculate their values based on the values returned by the *Position Class*. The attributes are handled as a combination by the *Attributes Class*. The *Position Class* computes all the distance values relying on the positions of the figures on the board. The *Hash Class* contains statistical functions which test the properties of the new computed database. The *Main File* reads the primary database and calls functions from the other classes, depending on what the *Option Class*, which deals with the command line options, specifies. Some *Utilities* are also included in the script, as well as *Another Main File* to test the attributes.

Attribute Classes

```
1  ##############################################################################
2  # Pieces files and rank
3  ##
4
5  # white king
6
7  class Attr_WK_file
8    def getHeader
9      return "@attribute whiteKingFile {a, b, c, d, e, f, g, h}"
```

[1] http://www.ruby-lang.org/

```
10      end
11
12      def compute(position)
13        return position.wkf.chr
14      end
15
16      def name
17        return "white king file"
18      end
19    end
20
21    class Attr_WK_file_Num
22      def getHeader
23        return "@attribute whiteKingFile numeric"
24      end
25
26      def compute(position)
27        return position.wkf - "a"[0] + 1
28      end
29
30      def name
31        return "white king file as a number"
32      end
33    end
34
35    class Attr_WK_rank
36      def getHeader
37        return "@attribute whiteKingRank {1, 2, 3, 4, 5, 6, 7, 8}"
38      end
39
40      def compute(position)
41        return position.wkr.chr
42      end
43
44      def name
45        return "white king rank"
46      end
47    end
48
49    class Attr_WK_rank_Num
50      def getHeader
51        return "@attribute whiteKingRank numeric"
52      end
53
54      def compute(position)
55        return position.wkr.chr.to_i
56      end
```

```
57
58      def name
59        return "white king rank as a number"
60      end
61    end
62
63    # white rook
64
65    class Attr_WR_file
66      def getHeader
67        return "@attribute whiteRookFile {a, b, c, d, e, f, g, h}"
68      end
69
70      def compute(position)
71        return position.wrf.chr
72      end
73
74      def name
75        return "white rook file"
76      end
77    end
78
79    class Attr_WR_file_Num
80      def getHeader
81        return "@attribute whiteRookFile numeric"
82      end
83
84      def compute(position)
85        return position.wrf - "a"[0] + 1
86      end
87
88      def name
89        return "white rook file as a number"
90      end
91    end
92
93    class Attr_WR_rank
94      def getHeader
95        return "@attribute whiteRookRank {1, 2, 3, 4, 5, 6, 7, 8}"
96      end
97
98      def compute(position)
99        return position.wrr.chr
100     end
101
102     def name
103       return "white rook rank"
```

```
104     end
105   end
106
107   class Attr_WR_rank_Num
108     def getHeader
109       return "@attribute whiteRookRank numeric"
110     end
111
112     def compute(position)
113       return position.wrr.chr.to_i
114     end
115
116     def name
117       return "white rook rank as a number"
118     end
119   end
120
121   # black king
122
123   class Attr_BK_file
124     def getHeader
125       return "@attribute blackKingFile {a, b, c, d, e, f, g, h}"
126     end
127
128     def compute(position)
129       return position.bkf.chr
130     end
131
132     def name
133       return "black king file"
134     end
135   end
136
137   class Attr_BK_file_Num
138     def getHeader
139       return "@attribute blackKingFile numeric"
140     end
141
142     def compute(position)
143       return position.bkf - "a"[0] + 1
144     end
145
146     def name
147       return "black king file as a number"
148     end
149   end
150
```

```
151  class Attr_BK_rank
152    def getHeader
153      return "@attribute blackKingRank {1, 2, 3, 4, 5, 6, 7, 8}"
154    end
155
156    def compute(position)
157      return position.bkr.chr
158    end
159
160    def name
161      return "black king rank"
162    end
163  end
164
165  class Attr_BK_rank_Num
166    def getHeader
167      return "@attribute blackKingRank numeric"
168    end
169
170    def compute(position)
171      return position.bkr.chr.to_i
172    end
173
174    def name
175      return "black king rank as a number"
176    end
177  end
178
179
180  ################################################################################
181  # depth of win
182  ##
183
184  class Attr_Depth_of_win
185    def getHeader
186      return "@attribute depth_of_win {draw, zero, one, two, three, four, five, six,
187    seven, eight, nine, ten, eleven, twelve, thirteen, fourteen, fifteen, sixteen}"
188    end
189
190    def compute(position)
191      return position.dow
192    end
193
194    def name
195      return "depth of win"
196    end
197  end
```

```
198
199   class Attr_Depth_of_win_Num
200     def getHeader
201       return "@attribute depth_of_win_num numeric"
202     end
203
204     def compute(position)
205       return {"draw"=>17, "zero"=>0, "one"=>1, "two"=>2, "three"=>3, "four"=>4,
206          "five"=>5, "six"=>6, "seven"=>7, "eight"=>8, "nine"=>9, "ten"=>10,
207          "eleven"=>11, "twelve"=>12, "thirteen"=>13, "fourteen"=>14,
208          "fifteen"=>15, "sixteen"=>16}[position.dow]
209     end
210
211     def name
212       return "depth of win as number"
213     end
214   end
215
216
217   ################################################################################
218   # attributes: Adjacents
219   ##
220
221   # black king threats white rook
222   class Attr_Adjacent_WR_BK
223     def getHeader
224       return "@attribute adjacent_WR_BK {true, false}"
225     end
226
227     def compute(position)
228         return(position.dist_WR_BK == 1)
229     end
230
231     def name
232       return "the black king threats the white rook"
233     end
234   end
235
236   # white king protects white rook
237   class Attr_Adjacent_WK_WR
238     def getHeader
239       return "@attribute adjacent_WK_WR {true, false}"
240     end
241
242     def compute(position)
243         return(position.dist_WK_WR == 1)
244     end
```

```
245
246     def name
247       return "the white king protects the white rook"
248     end
249   end
250
251   # the two kings are adjacent: illegal
252   class Attr_Adjacent_WK_BK
253     def getHeader
254       return "@attribute adjacent_WK_BK {true, false}"
255     end
256
257     def compute(position)
258         return(position.dist_WK_BK == 1)
259     end
260
261     def name
262       return "the two kings are on adjacent squares"
263     end
264   end
265
266   ##########################################################################
267   # attributes: Between
268   ##
269
270   # white rook divides the kings
271   class Attr_WR_between_WK_BK
272     def getHeader
273       return "@attribute WR_between_WK_BK {true, false}"
274     end
275
276     def compute(position)
277       return( smallerThan(position.wkf, position.wrf, position.bkf) or
278           smallerThan(position.wkr, position.wrr, position.bkr) or
279           smallerThan(position.bkf, position.wrf, position.wkf) or
280           smallerThan(position.bkr, position.wrr, position.wkr) )
281     end
282
283     def name
284       return "the white rook divides the both kings"
285     end
286   end
287
288   # white king divides white rook and black king
289   class Attr_WK_between_WR_BK
290     def getHeader
291       return "@attribute WK_between_WR_BK {true, false}"
```

```
292     end
293
294     def compute(position)
295       return(smallerThan(position.wrf, position.wkf, position.bkf) or
296                smallerThan(position.wrr, position.wkr, position.bkr) or
297                smallerThan(position.bkf, position.wkf, position.wrf) or
298                smallerThan(position.bkr, position.wkr, position.wrr))
299     end
300
301     def name
302       return "the order of the pieces is white rook-white king-black king"
303     end
304   end
305
306   # black king divides white king and white rook
307   class Attr_BK_between_WK_WR
308     def getHeader
309       return "@attribute BK_between_WK_WR {true, false}"
310     end
311
312     def compute(position)
313       return(smallerThan(position.wkf, position.bkf, position.wrf) or
314          smallerThan(position.wkr, position.bkr, position.wrr) or
315          smallerThan(position.wrf, position.bkf, position.wkf) or
316          smallerThan(position.wrr, position.bkr, position.wkr))
317     end
318
319     def name
320       return "the order of the pieces is white king-black king-white rook"
321     end
322   end
323
324
325   ##############################################################################
326   # attributes: Pieces distances
327   ##
328
329   # distance between the kings
330   class Attr_Distance_WK_BK
331     def getHeader
332       return "@attribute distance_WK_BK {1, 2, 3, 4, 5, 6, 7}"
333     end
334
335     def compute(position)
336       return position.dist_WK_BK
337     end
338
```

```
339     def name
340       return "the distance between the kings is"
341     end
342   end
343
344   # distance between the kings (numeric)
345   class Attr_Distance_WK_BK_Num
346     def getHeader
347       return "@attribute distance_WK_BK_num numeric"
348     end
349
350     def compute(position)
351       return position.dist_WK_BK
352     end
353
354     def name
355       return "the distance between the kings as number is"
356     end
357   end
358
359   # distance between white rook and black king
360   class Attr_Distance_WR_BK
361     def getHeader
362       return "@attribute distance_WR_BK {1, 2, 3, 4, 5, 6, 7}"
363     end
364
365     def compute(position)
366       return position.dist_WR_BK
367     end
368
369     def name
370       return "the distance between the white rook and the black king is"
371     end
372   end
373
374   # distance between white rook and black king (numeric)
375   class Attr_Distance_WR_BK_Num
376     def getHeader
377       return "@attribute distance_WR_BK_num numeric"
378     end
379
380     def compute(position)
381       return position.dist_WR_BK
382     end
383
384     def name
385       return "the distance between the white rook and the black king as a number is"
```

```
386     end
387   end
388
389   # distance between white king and white rook
390   class Attr_Distance_WK_WR
391     def getHeader
392       return "@attribute distance_WK_WR {0, 1, 2, 3, 4, 5, 6, 7}"
393     end
394
395     def compute(position)
396       return position.dist_WK_WR
397     end
398
399     def name
400       return "the distance between the white king and the white rook is"
401     end
402   end
403
404   # distance between white king and white rook (numeric)
405   class Attr_Distance_WK_WR_Num
406     def getHeader
407       return "@attribute distance_WK_WR_num numeric"
408     end
409
410     def compute(position)
411       return position.dist_WK_WR
412     end
413
414     def name
415       return "the distance between the white king and the white rook as a number is"
416     end
417   end
418
419   ##########################################################################
420   # attributes: Pieces file and rank distances
421   ##
422
423   # file distance between the kings
424   class Attr_File_distance_WK_BK
425     def getHeader
426       return "@attribute file_distance_WK_BK {0, 1, 2, 3, 4, 5, 6, 7}"
427     end
428
429     def compute(position)
430       return position.fileDist_WK_BK
431     end
432
```

```
433     def name
434       return "the file distance between the kings is"
435     end
436   end
437
438   # file distance between the kings (numeric)
439   class Attr_File_distance_WK_BK_Num
440     def getHeader
441       return "@attribute file_distance_WK_BK_num numeric"
442     end
443
444     def compute(position)
445       return position.fileDist_WK_BK
446     end
447
448     def name
449       return "the file distance between the kings as number is"
450     end
451   end
452
453   # rank distance between the kings
454   class Attr_Rank_distance_WK_BK
455     def getHeader
456       return "@attribute rank_distance_WK_BK {0, 1, 2, 3, 4, 5, 6, 7}"
457     end
458
459     def compute(position)
460       return position.rankDist_WK_BK
461     end
462
463     def name
464       return "the rank distance between the kings is"
465     end
466   end
467
468   # rank distance between the kings (numeric)
469   class Attr_Rank_distance_WK_BK_Num
470     def getHeader
471       return "@attribute rank_distance_WK_BK_num numeric"
472     end
473
474     def compute(position)
475       return position.rankDist_WK_BK
476     end
477
478     def name
479       return "the rank distance between the kings as number is"
```

```
480     end
481   end
482
483   # file distance between white rook and black king
484   class Attr_File_distance_WR_BK
485     def getHeader
486       return "@attribute file_distance_WR_BK {0, 1, 2, 3, 4, 5, 6, 7}"
487     end
488
489     def compute(position)
490       return position.fileDist_WR_BK
491     end
492
493     def name
494       return "the file distance between the white rook and the black king is"
495     end
496   end
497
498   # file distance between white rook and black king (numeric)
499   class Attr_File_distance_WR_BK_Num
500     def getHeader
501       return "@attribute file_distance_WR_BK_num numeric"
502     end
503
504     def compute(position)
505       return position.fileDist_WR_BK
506     end
507
508     def name
509       return "the file distance between the white rook and the black king as a number is"
510     end
511   end
512
513   # rank distance between white rook and black king
514   class Attr_Rank_distance_WR_BK
515     def getHeader
516       return "@attribute rank_distance_WR_BK {0, 1, 2, 3, 4, 5, 6, 7}"
517     end
518
519     def compute(position)
520       return position.rankDist_WR_BK
521     end
522
523     def name
524       return "the rank distance between the white rook and the black king is"
525     end
526   end
```

```
527
528   # rank distance between white rook and black king (numeric)
529   class Attr_Rank_distance_WR_BK_Num
530     def getHeader
531       return "@attribute rank_distance_WR_BK_num numeric"
532     end
533
534     def compute(position)
535       return position.rankDist_WR_BK
536     end
537
538     def name
539       return "the rank distance between the white rook and the black king as a number is"
540     end
541   end
542
543   # file distance between white king and white rook
544   class Attr_File_distance_WK_WR
545     def getHeader
546       return "@attribute file_distance_WK_WR {0, 1, 2, 3, 4, 5, 6, 7}"
547     end
548
549     def compute(position)
550       return position.fileDist_WK_WR
551     end
552
553     def name
554       return "the file distance between the white king and the white rook is"
555     end
556   end
557
558   # file distance between white king and white rook (numeric)
559   class Attr_File_distance_WK_WR_Num
560     def getHeader
561       return "@attribute file_distance_WK_WR_num numeric"
562     end
563
564     def compute(position)
565       return position.fileDist_WK_WR
566     end
567
568     def name
569       return "the file distance between the white king and the white rook as a number is"
570     end
571   end
572
573   # rank distance between white king and white rook
```

```
574  class Attr_Rank_distance_WK_WR
575    def getHeader
576      return "@attribute rank_distance_WK_WR {0, 1, 2, 3, 4, 5, 6, 7}"
577    end
578
579    def compute(position)
580      return position.rankDist_WK_WR
581    end
582
583    def name
584      return "the rank distance between the white king and the white rook is"
585    end
586  end
587
588  # rank distance between white king and white rook (numeric)
589  class Attr_Rank_distance_WK_WR_Num
590    def getHeader
591      return "@attribute rank_distance_WK_WR_num numeric"
592    end
593
594    def compute(position)
595      return position.rankDist_WK_WR
596    end
597
598    def name
599      return "the rank distance between the white king and the white rook as a number is"
600    end
601  end
602
603  ##############################################################################
604  # attributes: Board distances
605  ##
606
607  # distance from the black king to the closest edge
608  class Attr_Distance_BK_ce
609    def getHeader
610      return "@attribute Distance_WK_ce {0, 1, 2, 3}"
611    end
612
613    def compute(position)
614      return position.dist_BK_ce
615    end
616
617    def name
618      return "the distance from the black king to the closest edge is"
619    end
620  end
```

```
621
622   # distance from the black king to the closest edge (numeric)
623   class Attr_Distance_BK_ce_Num
624     def getHeader
625       return "@attribute Distance_WK_ce_num numeric"
626     end
627
628     def compute(position)
629       return position.dist_BK_ce
630     end
631
632     def name
633       return "the distance from the black king to the closest edge as a number is"
634     end
635   end
636
637   # distance of the black king to the closest corner
638   class Attr_Distance_BK_co
639     def getHeader
640       return "@attribute Distance_BK_ce {0, 1, 2, 3}"
641     end
642
643     def compute(position)
644       return position.dist_BK_co
645     end
646
647     def name
648       return "the distance from the black king to the closest corner is"
649     end
650   end
651
652   class Attr_Distance_BK_co_Num
653     def getHeader
654       return "@attribute Distance_BK_ce_num numeric"
655     end
656
657     def compute(position)
658       return position.dist_BK_co
659     end
660
661     def name
662       return "the distance from the black king to the closest corner as a number is"
663     end
664   end
665
666   # distance of the white to the central cross
667   class Attr_Distance_WK_cr
```

```
668     def getHeader
669       return "@attribute Distance_WK_cc {0, 1, 2, 3}"
670     end
671
672     def compute(position)
673       return position.dist_WK_cr
674     end
675
676     def name
677       return "the distance from the white king to the central cross is"
678     end
679   end
680
681   class Attr_Distance_WK_cr_Num
682     def getHeader
683       return "@attribute Distance_WK_cc_num numeric"
684     end
685
686     def compute(position)
687       return position.dist_WK_cr
688     end
689
690     def name
691       return "the distance from the white king to the central cross as a number is"
692     end
693   end
694
695   # distance of the white to the closest edge
696   class Attr_Distance_WK_ce
697     def getHeader
698       return "@attribute Distance_WK_ce {0, 1, 2, 3}"
699     end
700
701     def compute(position)
702       return position.dist_WK_ce
703     end
704
705     def name
706       return "the distance from the white king to the closest edge is"
707     end
708   end
709
710   class Attr_Distance_WK_ce_Num
711     def getHeader
712       return "@attribute Distance_WK_ce_num numeric"
713     end
714
```

```
715     def compute(position)
716       return position.dist_WK_ce
717     end
718
719     def name
720       return "the distance from the white king to the closest edge as a number is"
721     end
722   end
723
724   # distance of the white to the closest corner
725   class Attr_Distance_WK_co
726     def getHeader
727       return "@attribute Distance_WK_co {0, 1, 2, 3}"
728     end
729
730     def compute(position)
731       return position.dist_WK_co
732     end
733
734     def name
735       return "the distance from the white king to the closest corner is"
736     end
737   end
738
739   class Attr_Distance_WK_co_Num
740     def getHeader
741       return "@attribute Distance_WK_co_num numeric"
742     end
743
744     def compute(position)
745       return position.dist_WK_co
746     end
747
748     def name
749       return "the distance from the white king to the closest corner is"
750     end
751   end
752
753   ##############################################################################
754   # attributes: oriented caracteristics
755   ##
756
757   # Is the vertical distance between WR and BBK equal 1?
758   class Attr_Vertical_distance_WR_BK
759     def getHeader
760       return "@attribute vertical_distance_WR_BK_1 {not_one_square, one_square}"
761     end
```

```
762
763     def compute(position)
764       choosenBottom = position.bottom
765
766       case choosenBottom
767       when "a"
768         if position.wrf == position.bkf + 1 then return "one_square"
769         else return "not_one_square" end
770       when "h"
771         if position.wrf == position.bkf - 1 then return "one_square"
772         else return "not_one_square" end
773       when "1"
774         if position.wrr == position.bkr + 1 then return "one_square"
775         else return "not_one_square" end
776       when "8"
777         if position.wrr == position.bkr - 1 then return "one_square"
778         else return "not_one_square" end
779       end
780     end
781
782     def name
783       return "the vertical distance (1) between the black king and the rook is"
784     end
785   end
786
787   # Is the vertical distance between WK and BK equal 2?
788   class Attr_Vertical_distance_WK_BK
789     def getHeader
790       return "@attribute vertical_distance_WK_BK_1 {not_two_squares, two_squares}"
791     end
792
793     def compute(position)
794       choosenBottom = position.bottom
795
796       case choosenBottom
797       when "a"
798         if position.wkf == position.bkf + 2 then return "two_squares"
799         else return "not_two_squares" end
800       when "h"
801         if position.wkf == position.bkf - 2 then return "two_squares"
802         else return "not_two_squares" end
803       when "1"
804         if position.wkr == position.bkr + 2 then return "two_squares"
805         else return "not_two_squares" end
806       when "8"
807         if position.wkr == position.bkr - 2 then return "two_squares"
808         else return "not_two_squares" end
```

```
809       end
810     end
811
812     def name
813       return "the vertical distance between the both kings is"
814     end
815   end
816
817   # How is the horizontal distance between WK and BK
818   class Attr_Horizontal_distance_WK_BK
819     def getHeader
820       return "@attribute horizontal_distance_WK_BK_1 {odd, even, zero}"
821     end
822
823     def compute(position)
824       choosenBottom = position.bottom
825
826       case choosenBottom
827       when "a", "h"
828         if position.rankDist_WK_BK == 0
829           return "zero"
830         elsif position.rankDist_WK_BK % 2 == 0
831           return "even"
832         else
833           return "odd"
834         end
835
836       when "1", "8"
837         if position.fileDist_WK_BK == 0
838           return "zero"
839         elsif position.fileDist_WK_BK % 2 == 0
840           return "even"
841         else
842           return "odd"
843         end
844
845       end
846     end
847
848     def name
849       return "the horizontal distance between the both kings is"
850     end
851   end
852
853
854   ##############################################################################
855   # attributes: special positions
```

```
856   ##
857
858   # kings are in opposition
859   class Attr_K_in_opp
860     def getHeader
861       return "@attribute K_in_opp {true, false}"
862     end
863
864     def compute(position)
865       return( (position.fileDist_WK_BK == 0 and position.rankDist_WK_BK == 2) or
866             (position.fileDist_WK_BK == 2 and position.rankDist_WK_BK == 0) )
867     end
868
869     def name
870       return "the kings are in opposition"
871     end
872   end
873
874   # kings almost in opposition
875   class Attr_K_almost_in_opp
876     def getHeader
877       return "@attribute K_almost_in_opp {true, false}"
878     end
879
880     def compute(position)
881       return( (position.fileDist_WK_BK == 2 and position.rankDist_WK_BK == 1) or
882         or (position.fileDist_WK_BK == 1 and position.rankDist_WK_BK == 2) )
883     end
884
885     def name
886       return "the both kings are almost in opposition"
887     end
888   end
889
890   # white king, black king and white rook form a 'L' pattern
891   class Attr_L_patt
892     def getHeader
893       return "@attribute L_patt {true, false}"
894     end
895
896     def compute(position)
897       return( (position.fileDist_WK_BK == 0 and position.rankDist_WR_BK == 0 and
898          position.rankDist_WK_BK == 2) or (position.rankDist_WK_BK == 0 and
899         position.fileDist_WR_BK == 0 and position.fileDist_WK_BK == 2))
900     end
901
902     def name
```

```
903       return "there is a L-pattern"
904     end
905   end
906
907
908   ################################################################################
909   # attributes: special positions on the board
910   ##
911
912   # the white rook divides the 2 kings toward the closest edge
913   class Attr_WR_divides_K_toward_ce
914     def getHeader
915       return "@attribute WR_divides_K_toward_ce {true, false}"
916     end
917
918     def compute(position)
919       return ((smallerThan(position.wkf, position.wrf, position.bkf) and
920                    position.dist_BK_h == position.dist_BK_ce) or
921                  (smallerThan(position.wkr, position.wrr, position.bkr) and
922                   position.dist_BK_8 == position.dist_BK_ce) or
923                  (smallerThan(position.bkf, position.wrf, position.wkf) and
924                   position.dist_BK_a == position.dist_BK_ce) or
925                  (smallerThan(position.bkr, position.wrr, position.wkr) and
926                   position.dist_BK_1 == position.dist_BK_ce))
927     end
928
929     def name
930       return "the white rook divides the two kings towards the closest edge"
931     end
932   end
933
934   # the white rook holds the black king towards the closest edge
935   class Attr_WR_holds_BK_toward_ce
936     def getHeader
937       return "@attribute WR_holds_BK_toward_ce {true, false}"
938     end
939
940     def compute(position)
941       return
942     ((position.wrf == position.bkf + 1 and position.bk_closest_edge.include? "a") or
943      (position.wrf == position.bkf - 1 and position.bk_closest_edge.include? "h") or
944      (position.wrr == position.bkf + 1 and position.bk_closest_edge.include? "1") or
945      (position.wrr == position.bkr - 1 and position.bk_closest_edge.include? "8"))
946     end
947
948     def name
949       return "the white rook holds the black king towards the closest edge"
```

```
950     end
951   end
952
953   # the kings are in opposition that forces the black king towards the closest edge
954   class Attr_K_in_opp_toward_ce
955     def getHeader
956       return "@attribute K_in_opp_toward_ce {true, false}"
957     end
958
959     def compute(position)
960       return ((position.wkf == position.bkf and position.wkr == position.bkr + 2 and
961     position.bk_closest_edge.include? "1") or (position.wkf == position.bkf and
962     position.wkr == position.bkr - 2 and position.bk_closest_edge.include? "8") or
963    (position.wkf == position.bkf + 2 and position.wkr == position.bkr and
964     position.bk_closest_edge.include? "a") or (position.wkf == position.bkf - 2 and
965     position.wkr == position.bkr and position.bk_closest_edge.include? "h"))
966     end
967
968     def name
969       return "the both kings are in opposition " +
970        "so that the black king is held towards the closest edge"
971     end
972   end
973
974   # the white rook divides the 2 kings toward the closest edge and
975   #  the it holds the black king toward the closest edge
976   class Attr_WR_squeeze_BK
977     def getHeader
978       return "@attribute WR_squeeze_BK {true, false}"
979     end
980
981     def compute(position)
982       a = Attr_WR_divides_K_toward_ce.new
983       b = Attr_K_in_opp_toward_ce.new
984       return (a.compute(position) and b.compute(position))
985     end
986
987     def name
988       return "the black king is squeezed towards the closest edge"
989     end
990   end
991
992   ##########################################################################
993   # attributes: Other
994   ##
995
996   # free space around the black king as let free by the rook
```

```
997  class Attr_BK_free_space
998    def getHeader
999      return "@attribute BK_free_space {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14,
1000       15, 16, 18, 20, 21, 24, 25, 28, 30, 35, 36, 42, 49}"
1001   end
1002
1003   def compute(position)
1004     return position.rook_free_space
1005   end
1006
1007   def name
1008     return "the free space of the black king as let by the rook is"
1009   end
1010 end
1011
1012 class Attr_BK_free_space_Num
1013   def getHeader
1014     return "@attribute BK_free_space_num numeric"
1015   end
1016
1017   def compute(position)
1018     return position.rook_free_space
1019   end
1020
1021   def name
1022     return "the free space of the black king as let by the rook as a number is"
1023   end
1024 end
1025
1026 # true free space around the black king
1027 class Attr_BK_true_free_space
1028   def getHeader
1029     return "@attribute BK_true_free_space {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
1030 12, 14, 15, 16, 18, 20, 21, 24, 25, 28, 30, 35, 36, 42, 49}"
1031   end
1032
1033   def compute(position)
1034     return position.free_space
1035   end
1036
1037   def name
1038     return "the number of fields where the black king can go"
1039   end
1040 end
1041
1042 class Attr_BK_true_free_space_Num
1043   def getHeader
```

```
1044      return "@attribute BK_true_free_space_num numeric"
1045    end
1046
1047    def compute(position)
1048      return position.free_space
1049    end
1050
1051    def name
1052      return "the number of fields where the black king can go"
1053    end
1054  end
1055
1056  # WR and BK are in the "same zone"
1057  class Attr_Same_zone_WR_BK_1
1058    def getHeader
1059      return "@attribute same_zone_WR_BK_1 {true, false}"
1060    end
1061
1062    def compute(position)
1063      choosenBottom = position.bottom
1064
1065      case choosenBottom
1066      when "a", "h"
1067        return ((position.wrr <= "3"[0] and position.bkr <= "3"[0]) or
1068             (position.wrr >= "3"[0] and position.wrr <= "6"[0] and
1069             position.bkr >= "3"[0] and position.bkr <= "6"[0]) or
1070            (position.wrr >= "6"[0] and position.bkr >= "6"[0]))
1071      when "1", "8"
1072        return ((position.wrf <= "c"[0] and position.bkf <= "c"[0]) or
1073         (position.wrf >= "c"[0] and position.wrf <= "f"[0] and
1074          position.bkf >= "c"[0] and position.bkf <= "f"[0]) or
1075        (position.wrf >= "f"[0] and position.bkf >= "f"[0]))
1076      end
1077    end
1078
1079    def name
1080      return "the black king and the rook are in the same zone: "
1081    end
1082  end
1083
1084  ##########################################################################
1085  # attributes: mine
1086  ##
1087
1088  # file or rank WR - BK distance
1089  class Attr_Oriented_distance_WR_BK
1090    def getHeader
```

```
1091      return "@attribute oriented_dist_WR_BK {0,1,2,3,4,5,6,7}"
1092    end
1093
1094    def compute(position)
1095      return [position.fileDist_WR_BK, position.rankDist_WR_BK].min
1096    end
1097
1098    def name
1099      return "the distance between the black king and the file or the rank of the rook is "
1100    end
1101  end
1102
1103  # file or rank WR - BK distance num
1104  class Attr_Oriented_distance_WR_BK_Num
1105    def getHeader
1106      return "@attribute oriented_dist_WR_BK_num numeric"
1107    end
1108
1109    def compute(position)
1110      return [position.fileDist_WR_BK, position.rankDist_WR_BK].min
1111    end
1112
1113    def name
1114      return "the distance between the black king and the file or the rank of the rook is "
1115    end
1116  end
1117
1118
1119
```

Attributes Class

```
1  class Attributes
2    attr_accessor :list, :mode
3    def initialize(mode = "enum")
4      @list = Array.new
5      @mode = mode
6    end
7
8  # add one or more attributes to the list
9    def add(*attrArray)
10     attrArray.each do |at|
11       if at.class == Array then
12         at.each { |att| self.addOne(att) }
13       else
```

```
14            self.addOne(at)
15          end
16        end
17      end
18
19      def addOne(attr)
20        if @mode == "numeric" and
21            Module.constants.include?(attr.name.to_s + "_Num")
22          @list.push Module.const_get(attr.name.to_s + "_Num").new
23        else
24          @list.push attr.new
25        end
26      end
27
28    #return the header line for all the attributes
29      def getHeader
30        ret = ""
31        @list.each { |attr| ret += attr.getHeader + "\n"}
32        ret += Attr_Depth_of_win.new.getHeader + "\n"
33        return ret
34      end
35
36      def values(position)
37        vals = Array.new
38        @list.each { |attr| vals.push attr.compute(position).to_s }
39        return vals
40      end
41
42      def names
43        vals = Array.new
44        @list.each { |attr| vals.push attr.name }
45        return vals
46      end
47
48      def summary(position)
49        vals = Array.new
50        @list.each { |attr|
51          vals.push attr.name + ": " + attr.compute(position).to_s
52        }
53        return vals
54      end
55    end
```

Position Class

```
1  class Position
2    attr_accessor :wkf, :wkr, :wrf, :wrr, :bkf, :bkr, :dow
3    def initialize
4      @wkf = @wkr = @wrf = @wrr = @bkf = @bkr = 0
5      @dow = ""
6    end
7
8    # define the position with the 7 first elements of the array
9    def readArray(array)
10     if array.class == Array and array.size >= 7 then
11       @wkf = array[0][0]; @wkr = array[1][0]
12       @wrf = array[2][0]; @wrr = array[3][0]
13       @bkf = array[4][0]; @bkr = array[5][0]
14       @dow = array[6]
15     else
16       puts array.inspect
17       puts "Position::readArray() unvalid argument"
18     end
19   end
20
21   def to_s
22     @wkf.chr + @wkr.chr + "," + @wrf.chr + @wrr.chr +
23        "," + @bkf.chr + @bkr.chr
24   end
25 ##########################################################################
26 # all distances
27 ##
28
29   def dist(a, b)
30     case a.downcase
31       when "wk" then return self.dist_WK(b)
32       when "wr" then return self.dist_WR(b)
33       when "bk" then return self.dist_BK(b)
34     end
35   end
36
37   def dist_WK(a)
38     case a.downcase
39       when "wk" then return 0
40       when "wr" then return self.dist_WK_WR
41       when "bk" then return self.dist_WK_BK
42       when "a" then return self.dist_WK_a
43       when "h" then return self.dist_WK_h
44       when "1" then return self.dist_WK_1
```

```
45          when "8" then return self.dist_WK_8
46          when "ce" then return self.dist_WK_ce
47          when "cr" then return self.dist_WK_cr
48          when "co" then return self.dist_WK_co
49        end
50      end
51
52      def dist_WR(a)
53        case a.downcase
54          when "wk" then return self.dist_WK_WR
55          when "wr" then return 0
56          when "bk" then return self.dist_WR_BK
57          when "a" then return self.dist_WR_a
58          when "h" then return self.dist_WR_h
59          when "1" then return self.dist_WR_1
60          when "8" then return self.dist_WR_8
61          when "ce" then return self.dist_WR_ce
62          when "cr" then return self.dist_WR_cr
63          when "co" then return self.dist_WR_co
64        end
65      end
66
67      def dist_BK(a)
68        case a.downcase
69          when "wk" then return self.dist_WK_BK
70          when "wr" then return self.dist_WR_BK
71          when "bk" then return 0
72          when "a" then return self.dist_BK_a
73          when "h" then return self.dist_BK_h
74          when "1" then return self.dist_BK_1
75          when "8" then return self.dist_BK_8
76          when "ce" then return self.dist_BK_ce
77          when "cr" then return self.dist_BK_cr
78          when "co" then return self.dist_BK_co
79        end
80      end
81
82    ##########################################################################
83    # file distances between pieces
84    ##
85
86      def fileDist_WK_WR
87        return (@wkf - @wrf).abs
88      end
89
90      def fileDist_WK_BK
91        return (@wkf - @bkf).abs
```

```
92    end
93
94    def fileDist_WR_BK
95      return (@wrf - @bkf).abs
96    end
97
98  ##########################################################################
99  # rank distances between pieces
100 ##
101
102   def rankDist_WK_WR
103     return (@wkr - @wrr).abs
104   end
105
106   def rankDist_WK_BK
107     return (@wkr - @bkr).abs
108   end
109
110   def rankDist_WR_BK
111     return (@wrr - @bkr).abs
112   end
113
114 ##########################################################################
115 # distances between pieces
116 ##
117   def dist_WK_WR
118     return [self.fileDist_WK_WR, self.rankDist_WK_WR].max
119   end
120
121   def dist_WK_BK
122     return [self.fileDist_WK_BK, self.rankDist_WK_BK].max
123   end
124
125   def dist_WR_BK
126     return [self.fileDist_WR_BK, self.rankDist_WR_BK].max
127   end
128
129 ##########################################################################
130 # distances between WK and edges, central cross
131 ##
132
133   # first column
134   def dist_WK_a
135     return (@wkf - 'a'[0]).abs
136   end
137
138   #8th column
```

```
139     def dist_WK_h
140       return (@wkf - 'h'[0]).abs
141     end
142
143     #first row
144     def dist_WK_1
145       return (@wkr - '1'[0]).abs
146     end
147
148     # 8th row
149     def dist_WK_8
150       return (@wkr - '8'[0]).abs
151     end
152
153     #closest edge
154     def dist_WK_ce
155       return [self.dist_WK_a, self.dist_WK_h, self.dist_WK_1, self.dist_WK_8].min
156     end
157
158     #central cross
159     def dist_WK_cr
160       return [ [(@wkf - 'd'[0]).abs, (@wkf - 'e'[0]).abs].min,
161                     [(@wkr - '4'[0]).abs, (@wkr - '5'[0]).abs].min ].max
162     end
163
164     #closest corner
165     def dist_WK_co
166       return [ [(@wkf - 'a'[0]).abs, (@wkf - 'h'[0]).abs].min,
167                     [(@wkr - '1'[0]).abs, (@wkr - '8'[0]).abs].min ].max
168     end
169
170   ##########################################################################
171   # distances between WR and edges
172   ##
173
174     # first column
175     def dist_WR_a
176       return (@wrf - 'a'[0]).abs
177     end
178
179     #8th column
180     def dist_WR_h
181       return (@wrf - 'h'[0]).abs
182     end
183
184     #first row
185     def dist_WR_1
```

```
186       return (@wrr - '1'[0]).abs
187     end
188
189     # 8th row
190     def dist_WR_8
191       return (@wrr - '8'[0]).abs
192     end
193
194     #closest edge
195     def dist_WR_ce
196       return [self.dist_WR_a, self.dist_WR_h, self.dist_WR_1, self.dist_WR_8].min
197     end
198
199     #central cross
200     def dist_WR_cr
201       return [ [(@wrf - 'd'[0]).abs, (@wrf - 'e'[0]).abs].min,
202                     [(@wrr - '4'[0]).abs, (@wrr - '5'[0]).abs].min ].max
203     end
204
205     #closest corner
206     def dist_WR_co
207       return [ [(@wrf - 'a'[0]).abs, (@wrf - 'h'[0]).abs].min,
208                     [(@wrr - '1'[0]).abs, (@wrr - '8'[0]).abs].min ].max
209     end
210
211   ##########################################################################
212   # distances between BK and edges, corners
213   ##
214
215     # first column
216     def dist_BK_a
217       return (@bkf - 'a'[0]).abs
218     end
219
220     #8th column
221     def dist_BK_h
222       return (@bkf - 'h'[0]).abs
223     end
224
225     #first row
226     def dist_BK_1
227       return (@bkr - '1'[0]).abs
228     end
229
230     # 8th row
231     def dist_BK_8
232       return (@bkr - '8'[0]).abs
```

```
233     end
234
235     #closest edge
236     def dist_BK_ce
237       return [self.dist_BK_a, self.dist_BK_h, self.dist_BK_1, self.dist_BK_8].min
238     end
239
240     #closest corner
241     def dist_BK_co
242       return [ [self.dist_BK_a, self.dist_BK_h].min,
243                   [self.dist_BK_1, self.dist_BK_8].min ].max
244     end
245
246     #central cross
247     def dist_BK_cr
248       return [ [(@wrf - 'd'[0]).abs, (@wrf - 'e'[0]).abs].min,
249                   [(@wrr - '4'[0]).abs, (@wrr - '5'[0]).abs].min ].max
250     end
251
252   ##########################################################################
253   # closest edges
254   ##
255
256     # return an array with the WK's closest edge(s)
257     def wk_closest_edge
258       ce = Array.new
259       ce.push "a" if self.dist_WK_a == self.dist_WK_ce
260       ce.push "h" if self.dist_WK_h == self.dist_WK_ce
261       ce.push "1" if self.dist_WK_1 == self.dist_WK_ce
262       ce.push "8" if self.dist_WK_8 == self.dist_WK_ce
263       return ce
264     end
265
266     # return an array with the WR's closest edge(s)
267     def wr_closest_edge
268       ce = Array.new
269       ce.push "a" if self.dist_WR_a == self.dist_WR_ce
270       ce.push "h" if self.dist_WR_h == self.dist_WR_ce
271       ce.push "1" if self.dist_WR_1 == self.dist_WR_ce
272       ce.push "8" if self.dist_WR_8 == self.dist_WR_ce
273       return ce
274     end
275
276     # return an array with the BK's closest edge(s)
277     def bk_closest_edge
278       ce = Array.new
279       ce.push "a" if self.dist_BK_a == self.dist_BK_ce
```

```
280       ce.push "h" if self.dist_BK_h == self.dist_BK_ce
281       ce.push "1" if self.dist_BK_1 == self.dist_BK_ce
282       ce.push "8" if self.dist_BK_8 == self.dist_BK_ce
283       return ce
284     end
285
286   ##########################################################################
287   # vertical and horizontal detection
288   ##
289
290     def bottom
291       ces = self.bk_closest_edge
292       case ces.size
293       when 1 then return ces[0]
294       when 2
295         if self.dist_WK(ces[0]) <= self.dist_WK(ces[1])
296           return ces[0]
297         else
298           return ces[1]
299         end
300       end
301     end
302
303
304     def rook_free_space
305       if @wrf < @bkf then width = 'h'[0] - @wrf
306       elsif @wrf > @bkf then width = @wrf - 'a'[0]
307       else width = ['h'[0] - @wrf, @wrf - 'a'[0]].max end
308
309       if @wrr < @bkr then height = '8'[0] - @wrr
310       elsif @wrr > @bkr then height = @wrr - '1'[0]
311       else height = ['8'[0] - @wrr, @wrr - '1'[0]].max end
312
313       return(width * height)
314     end
315
316     def free_space
317       d = false
318
319
320       pos = @bkf.chr + @bkr.chr
321       board = ('a'..'h').to_a.collect{|a| ((a+'1')..(a+'8')).to_a}.flatten
322       grasp = ((@bkf-1).chr..(@bkf+1).chr).to_a.collect{ |a|
323         (a+(@bkr-1).chr..a+(@bkr+1).chr).to_a
324       }.flatten
325       wkgrasp = ((@wkf-1).chr..(@wkf+1).chr).to_a.collect{ |a|
326         (a+(@wkr-1).chr..a+(@wkr+1).chr).to_a
```

```
327       }.flatten
328
329       poss = (grasp & board) - Array[pos] - wkgrasp
330       fs = poss.size
331
332       for a in poss
333           i, j = a[0], a[1]
334           if i==@wrf and j!=@wrr and
335               !(j<@wkr and @wkr<@wrr and @wkf==@wrf) and
336               !(@wrr<@wkr and @wkr<j and @wkf==@wrf)
337
338             puts i.chr + "," + j.chr + ": rook file" if d; fs -= 1
339             next
340           end
341
342           if i!=@wrf and j==@wrr and
343               !(i<@wkf and @wkf<@wrf and @wkr==@wrr) and
344               !(@wrf<@wkf and @wkf<i and @wkr==@wrr)
345
346             puts i.chr + "," + j.chr + ": rook rank" if d
347             fs-=1
348             next
349           end
350
351           puts i.chr + "," + j.chr + " ok" if d
352       end
353       return fs
354     end
355
356   end
```

Hash Class

```
1   class MyHash
2     attr_reader :numberOfDifferentEntries, :numberOfEntries
3     attr_reader :numberOfAmbiguousEntries, :summary
4     attr_reader :hashContainer, :numbers
5
6     def initialize
7       @debug = false
8       @hashTrigger = false
9       @hashContainer = Hash.new
10      @hashContainerSummed = Hash.new
11      @dowContainer = Hash.new
12      @dowStatContainer = Hash.new
```

```
13      @dowContainerSummed = Hash.new
14
15      @numberOfDifferentEntries = 0
16      @numberOfEntries = 0
17      @numberOfAmbiguousEntries = 0
18      @summaries = Hash.new
19      @matrix1 = Array.new
20      @graph1 = Array.new
21      @numbers = ["draw", "zero","one","two", "three", "four",
22          "five", "six", "seven", "eight", "nine", "ten", "eleven",
23          "twelve", "thirteen", "fourteen", "fifteen", "sixteen"]
24    end
25
26    def addEntry(attributes, value)
27      if @hashTrigger
28        index = attributes.join.hash
29      else
30        index = attributes
31      end
32      @dowContainer[value] = Array.new if !@dowContainer[value]
33      @dowContainer[value].push attributes.join.hash
34      @hashContainer[index] = Array.new if !@hashContainer[index]
35      @hashContainer[index].push value
36
37      @dowContainerSummed[value] = Hash.new if !@dowContainerSummed[value]
38      @dowContainerSummed[value][index] = 0 if !@dowContainerSummed[value][index]
39      @dowContainerSummed[value][index] += 1
40      @hashContainerSummed[index] = Hash.new if !@hashContainerSummed[index]
41      @hashContainerSummed[index][value] = 0 if !@hashContainerSummed[index][value]
42      @hashContainerSummed[index][value] += 1
43    end
44
45    def myCompare(a, b)
46      a = a.sort {|x, y| @numbers.index(x[0]) <=> @numbers.index(y[0])}
47      b = b.sort {|x, y| @numbers.index(x[0]) <=> @numbers.index(y[0])}
48      puts "MyHash::myCompare(" + a.class.to_s + "," + b.class.to_s + ")" if @debug
49      puts "| comparing " + a.inspect + " and " + b.inspect + " ..." if @debug
50
51      if a.size > b.size then res = 1
52      elsif a.size < b.size then res = -1
53      else res = self.recurs(a, b)
54      end
55      puts "| result: " + res.to_s if @debug
56      puts "----------------------------------------" if @debug
57      return res
58    end
59
```

```
60      def recurs(a, b)
61        a = a.to_a if a.class == Hash
62        b = b.to_a if b.class == Hash
63        if @numbers.index(a[0][0]) > @numbers.index(b[0][0])
64          return 1
65        elsif @numbers.index(a[0][0]) < @numbers.index(b[0][0]) then return -1
66        else
67          if a.size == 1 then return 0
68          else return self.recurs(a[1..a.size], b[1..b.size])
69          end
70        end
71      end
72
73      def computeInfos
74        @numberOfDifferentEntries = @container.length
75        @numberOfEntries = 0
76        @numberOfAmbiguousEntries = 0
77        @hashContainer.each do |attributesSet, dowSet|
78          positionsOfTheSet = 0
79          summaryOfTheSet = ""
80          keyForSummaries = ("A"[0] + dowSet.length).chr
81
82          dowOfTheMax = @numbers.index(dowSet.max { |a, b| a[1] <=> b[1] }[0])
83          amplitudeOfTheSet = @numbers.index(dowSet.max{ |a, b|
84            @numbers.index(a[0]) <=> @numbers.index(b[0])
85          }[0]) - @numbers.index(dowSet.min{ |a, b|
86            @numbers.index(a[0]) <=> @numbers.index(b[0])
87          }[0])
88
89          @matrix1[dowOfTheMax] = Array.new if @matrix1[dowOfTheMax].class != Array
90          dowSet.sort{ |a, b| @numbers.index(a[0]) <=> @numbers.index(b[0]) }.each do
91            |dow, quantityOfPositions|
92
93            positionsOfTheSet += quantityOfPositions
94            summaryOfTheSet += (dow + "=>" + quantityOfPositions.to_s + ",")
95            keyForSummaries += ("A"[0] + @numbers.index(dow)).chr
96
97            if !@matrix1[dowOfTheMax][@numbers.index(dow)]
98              @matrix1[dowOfTheMax][@numbers.index(dow)] = quantityOfPositions
99            else
100             @matrix1[dowOfTheMax][@numbers.index(dow)] += quantityOfPositions
101           end
102
103           if !@graph1[@numbers.index(dow)] or
104              @graph1[@numbers.index(dow)] < amplitudeOfTheSet
105
106             @graph1[@numbers.index(dow)] = amplitudeOfTheSet
```

```
107           end
108
109         end
110         while @summaries.has_key? keyForSummaries do
111           keyForSummaries += "A"
112         end
113         @summaries[keyForSummaries] = "[" + attributesSet.join(",") +
114           "] => " + positionsOfTheSet.to_s +
115           " {" + summaryOfTheSet[0..-2] + "}"
116         @numberOfEntries += positionsOfTheSet
117         @numberOfAmbiguousEntries += positionsOfTheSet if dowSet.length > 1
118       end
119     end
120
121     def summary
122       summary = ""
123       print "computing the summary."
124       @hashContainerSummed.sort{ |a,b|
125         self.myCompare(a[1], b[1])
126       }.each {|attributes, values|
127         summary += "["
128
129         if @hashTrigger
130           summary += attributes.to_s
131         else
132           summary += attributes.join(",")
133         end
134
135         summary += "] => {"
136         values.sort{ |a,b|
137           @numbers.index(a[0]) <=> @numbers.index(b[0])
138         }.each { |dow, quantity|
139           summary += dow + ":" + quantity.to_s + ","
140         }
141         summary[summary.size-1] = '}'
142         summary += "\n"
143         print "." if @debug
144       }
145       print "\n"
146       return summary
147     end
148
149     def load(hash)
150       @hashContainer = hash
151     end
152
153     def amplitude
```

```
154       quartiles = true
155
156       if @debug
157         print "first loop size: ",
158             @dowContainerSummed.size,
159             "; second loop average size: ",
160             Kernel.eval( "(" +
161               @dowContainerSummed.to_a.collect { |a|
162                 a[1].size
163               }.join("+")+")/" +
164               @dowContainerSummed.size.to_s),
165             "; second loop iterations sum: ",
166             Kernel.eval("(" + @dowContainerSummed.to_a.collect{ |a|
167               a[1].size
168             }.join("+")+")"),
169             "\n"
170       end
171       print "calculating the amplitude."
172
173       graph = "#dow\tmin\tmed\tmax\tfirstQ\tthirdQ\tnb\tnbfalseNb\n"
174       @dowContainerSummed.sort{ |a,b|
175         @numbers.index(a[0]) <=> @numbers.index(b[0])
176       }.each { |dow, hashVals|
177         falseNb = 0
178         nb, min, max, sum, sum2 = 0, 10000000000, 0, 0, 0
179         allValues = Hash.new if quartiles
180         hashVals.each {|attrs, number|
181           @hashContainerSummed[attrs].each { |ndow, renumber|
182             nb += number * renumber
183             falseNb += renumber if dow != ndow
184             d = @numbers.index(ndow)
185             min = d if min > d
186             max = d if max < d
187             sum += d * renumber * number
188             sum2 += d**2 * number * renumber
189             if quartiles
190               allValues[ndow] = 0 if !allValues[ndow]
191               allValues[ndow] += number*renumber
192             end
193           }
194         }
195
196         size = 0
197         firstQ = 0
198         thirdQ = 0
199         if quartiles
200           allValues.each {|a, b| size += b}
```

```
201           fq = (size.to_f / 4).ceil
202           tq = (3 * size.to_f / 4).floor
203
204           allValues.sort{ |a,b|
205             @numbers.index(a[0]) <=> @numbers.index(b[0])
206           }.each{ |idow, quant|
207
208             fq -= quant
209             tq -= quant
210             firstQ = @numbers.index(idow) if fq <= 0 and firstQ == 0
211             thirdQ = @numbers.index(idow) if tq <= 0 and thirdQ == 0
212             break if fq < 0 and tq < 0
213           }
214         end
215         med = sum.to_f / nb
216         std = Math.sqrt((sum2.to_f / nb) - ((sum**2).to_f / (nb**2)))
217         print "."
218
219         graph += @numbers.index(dow).to_s + "\t" + min.to_s +
220         graph += "\t" + med.to_s + "\t" + max.to_s
221         if quartiles
222           graph += "\t" + firstQ.to_s + "\t" + thirdQ.to_s + "\t" + size.to_s
223         end
224         graph += "\t" + falseNb.to_s + "\n"
225       }
226       print "\n"
227       return graph
228     end
229
230     def graph2
231       graph = ""
232       for i in 0..17
233         graph += i.to_s + "\t" + @graph1[i].to_s + "\n"
234       end
235       return graph
236     end
237
238     def size
239       @hashContainer.size
240     end
241
242     def newDb
243       db = String.new
244       errors = Array.new
245       if @hashTrigger
246         raise "cannot print new database with hashed attributes"
247       end
```

```
248
249       @hashContainerSummed.sort{ |a,b|
250         self.myCompare(a[1], b[1])
251       }.each {|attributes, dows|
252         if dows.size == 1
253           db += attributes.join(",") + ',' + dows.keys[0] + "\n"
254         else
255           errors.push attributes
256         end
257       }
258       return db, errors
259     end
260   end
```

Main File

```
1   #!/usr/bin/ruby
2
3   require "utils.rb"
4   require "position.rb"
5   require "attributes.rb"
6   require "attribute.rb"
7   require "myhash.rb"
8   require "options.rb"
9
10  file = "./krkopt.data"
11  attributes = Attributes.new
12  options = Options.new
13  options.read ARGV
14
15  if options.attributes.length > 0 then
16    if options.numeric then attributes.mode = "numeric" end
17    attributes.add options.attributes
18  else
19    raise "error: no attribute given"
20  end
21
22  newdb = "@relation krk\n\n"
23  newdb << attributes.getHeader
24  newdb << "\n@data\n"
25
26  aFile = File.new(file, "r")
27  i = 0
28  hashTable = MyHash.new
29
```

```
30  t0 = Time.new
31  puts t0.strftime("[%H:%M:%S] calculating new db...")
32  aFile.each_line do |line|
33
34    if line.chomp.split(',').size < 7
35       raise "unvalid line " + i.to_s + ": " + line
36    end
37    position = Position.new
38    position.readArray line.chomp.split(',')
39
40    b = attributes.values(position)
41    if b.include? "irrelevant" then next end
42
43    hashTable.addEntry(b, position.dow)
44
45    i += 1
46    break if options.maxIterations > 0 and i >= options.maxIterations
47  end
48  aFile.close
49
50  puts Time.now.strftime("[%H:%M:%S]") + " " +
51          i.to_s + " lines calculated, analysing results..."
52
53  ##
54  # saving all produced lines
55  #
56  repartitionFile = File.new(options.resultsFileName + ".rep",
57          File::CREAT|File::TRUNC|File::WRONLY)
58  repartitionFile << "Stats\n "
59  repartitionFile << "size: " << hashTable.size << "\n"
60  repartitionFile << hashTable.summary
61  repartitionFile.close
62  puts "writing repartition into " + options.resultsFileName + ".rep"
63
64
65  ##
66  # saving the amplitudes for gnuplot to draw a graphic
67  ##
68  amplitudeFile = File.new(options.resultsFileName + ".amp",
69          File::CREAT|File::TRUNC|File::WRONLY)
70  amplitudeFile << hashTable.amplitude
71  amplitudeFile.close
72  puts "writing amplitudes into " + options.resultsFileName + ".amp"
73
74  ##
75  # new database
76  ##
```

```
77  dbFile = File.new(options.resultsFileName + ".arff",
78          File::CREAT|File::TRUNC|File::WRONLY)
79  dbFile << newdb
80  dbFile << (db, errors = hashTable.newDb; db)
81  dbFile.close
82  puts Time.now.strftime("[%H:%M:%S]") +
83          " writing new database into " +
84          options.resultsFileName + ".arff"
85  errorsFile = File.new(options.resultsFileName + ".dber",
86          File::CREAT|File::TRUNC|File::WRONLY)
87
88  aFile = File.new(file, "r")
89  aFile.each_line do |line|
90    if line.chomp.split(',').size < 7
91      raise "unvalid line " + i.to_s + ": " + line
92    end
93    position = Position.new
94    position.readArray line.chomp.split(',')
95
96    b = attributes.values(position)
97    if b.include? "irrelevant" then next end
98    if errors.include? b then errorsFile << line end
99    break if options.maxIterations > 0 and i >= options.maxIterations
100 end
101 aFile.close
102 errorsFile.close
103
104
105 puts Time.now.strftime("[%H:%M:%S]") +
106         " done after " + printTime(Time.new - t0)
```

Option Class

```
1  class Options
2    attr_reader :maxIterations, :attributes, :resultsFileName,
3    attr_reader :position, :numeric
4
5    def initialize
6      @maxIterations = 0
7      @attributes = Array.new
8      @resultsFileName = "results"
9      @position = Array.new
10     @numeric = false
11   end
12
```

```
13    def read(argv)
14      argv.each do |option|
15
16        # searching for option --max-lines=20 and -l=20
17        if option[0,12] == "--max-lines=" or option[0,3] == "-l=" then
18          _start = (0..option.size - 1).find {|x| option[x] == "="[0]} + 1
19          @maxIterations = option[_start..option.size].to_i
20          puts "reading only the " + @maxIterations.to_s +
21                 " first lines of the input file"
22
23        # searching for option --mode=morales and -m=s
24        elsif option[0,7] == "--mode=" or option[0,3] == "-m=" then
25          _start = (0..option.size - 1).find {|x| option[x] == "="[0]} + 1
26          _cut = (0..@resultsFileName.size - 1).find {|x|
27            @resultsFileName[x..@resultsFileName.length - 1] == ".arff"
28          }
29
30          case option[_start..option.size].downcase
31            when "m", "morales"
32              push("morales")
33              @resultsFileName = "morales"
34              puts "adding Morales' attributes"
35
36            when "t", "torres"
37              push "torres"
38              @resultsFileName = "torres"
39              puts "adding Torres y Quevedos' attributes"
40
41            when "b", "bain"
42               push "bain"
43              @resultsFileName = "bain"
44              puts "adding Bain's attributes"
45
46            when "s", "sadikov"
47              push "sadikov"
48              @resultsFileName = "sadikov"
49              puts "adding Sadikov's attributes"
50
51            when "a", "all"
52              push("bain"); push("morales"); push("sadikov")
53              push("torres"); push("gab")
54              @resultsFileName = "bmst"
55              puts "setting mode All (bmst)"
56
57            when "d", "debug"
58               push "debug"
59              @resultsFileName = "debug"
```

```
60              puts "adding debug's attributes"
61
62            when "g", "gab", "gabriel"
63              push "gab"
64              @resultsFileName = "gabriel"
65              puts "adding gabriel's special attributes"
66          end
67
68        # searching for option --out-file=results
69        elsif option[0,11] == "--out-file=" or option[0..2] == "-f=" then
70          _start = (0..option.size - 1).find {|x| option[x] == "="[0]} + 1
71          @resultsFileName = option[_start..option.size]
72          puts "setting results file name to " + @resultsFileName
73
74        # searching for option --position=a,1,b,2,c,3
75        elsif option[0,11] == "--position=" or option[0..2] == "-p=" then
76          _start = (0..option.size - 1).find {|x| option[x] == "="[0]} + 1
77          @position = option[_start..option.size].split(",")
78          @position.push ""
79
80  #       searching for option --numeric or -n
81        elsif option[0,9] == "--numeric" or option[0..1] == "-n" then
82          @numeric = true
83          puts "setting numeric on"
84        end
85      end
86    end
87
88    def push(name)
89      case name
90      when "bain"
91        @attributes.push(Attr_File_distance_WK_BK, Attr_Rank_distance_WK_BK,
92            Attr_File_distance_WR_BK, Attr_Rank_distance_WR_BK,
93            Attr_File_distance_WK_WR, Attr_Rank_distance_WK_WR,
94            Attr_WK_between_WR_BK, Attr_WR_between_WK_BK,
95            Attr_BK_between_WK_WR, Attr_Adjacent_WK_WR,
96            Attr_Adjacent_WK_BK, Attr_Adjacent_WR_BK)
97      when "morales"
98        @attributes.push(Attr_Adjacent_WR_BK, Attr_WR_between_WK_BK,
99            Attr_K_almost_in_opp, Attr_L_patt, Attr_WK_between_WR_BK)
100     when "sadikov"
101       @attributes.push(Attr_Distance_WK_BK, Attr_Distance_WR_BK,
102           Attr_K_in_opp, Attr_Distance_BK_ce, Attr_BK_free_space,
103           Attr_WR_between_WK_BK, Attr_WR_divides_K_toward_ce,
104           Attr_WR_holds_BK_toward_ce, Attr_K_in_opp_toward_ce,
105           Attr_WR_squeeze_BK, Attr_Distance_BK_co, Attr_Distance_WK_cr)
106     when "torres"
```

```
107         @attributes.push(Attr_Vertical_distance_WR_BK,
108             Attr_Same_zone_WR_BK_1, Attr_Vertical_distance_WK_BK,
109             Attr_Horizontal_distance_WK_BK)
110       when "gab"
111         @attributes.push(Attr_Distance_WR_BK, Attr_Distance_WK_WR,
112             Attr_Distance_WK_BK, Attr_Distance_BK_co,
113             Attr_Oriented_distance_WR_BK, Attr_BK_true_free_space,
114             Attr_BK_free_space, Attr_K_in_opp_toward_ce, Attr_Distance_WK_cr,
115             Attr_Distance_WK_co, Attr_K_almost_in_opp)
116       when "debug"
117         @attributes.push(Attr_WK_file, Attr_WK_rank, Attr_WR_file,
118             Attr_WR_rank, Attr_BK_file, Attr_BK_rank)
119       else
120         raise "no valable attributes group"
121       end
122       @attributes.uniq!
123     end
124   end
```

Utilities

```
1   def smallerThan(*int)
2     cond = true
3     if int.length > 1 then
4       for i in 0.. int.length - 2 do
5         cond = (cond and int[i] < int[i+1])
6       end
7     end
8     return cond
9   end
10
11  def printTime(timeInS)
12    if timeInS < 60.0
13      return timeInS.to_s + "s"
14    elsif timeInS < 3600.0
15      timeM = (timeInS / 60.0).floor
16      return timeM.to_s + "m" + (timeInS - 60.0 * timeM).to_s + "s"
17    elsif timeInS < 86400.0
18      timeH = (timeInS / 3600.0).floor
19      timeM = ((timeInS - 3600.0 * timeH) / 60.0).floor
20      return timeH.to_s + "h" + timeM.to_s + "m" +
21         (timeInS - 3600.0 * timeH - 60.0 * timeM).to_s + "s"
22    end
23  end
24
```

Another Main File to test the Attributes

```
1  #!/usr/bin/ruby
2
3  require "utils.rb"
4  require "position.rb"
5  require "attributes.rb"
6  require "attribute.rb"
7  require "attributetype.rb"
8  require "myhash.rb"
9  require "options.rb"
10
11 position = Position.new
12 attributes = Attributes.new
13 options = Options.new
14 options.read ARGV
15
16 attributes.add options.attributes
17
18 if options.position.empty? then
19   puts "error: no position given to test attributes"
20   exit
21 else
22   position.readArray options.position
23   i = 0
24   attributes.summary(position).each { |a| print i, "-", a,"\n"; i+= 1}
25   puts
26 end
27
28
```

Bibliography

[Bai91] M. Bain. Experiments in non-monotonic learning. In L. Birnbaum and G. Collins, editors, *Proceedings of the eighth International Workshop on Machine Learning*, pages 380–384. Morgan Kaufmann, San Mateo, CA, 1991.

[Bai94] M. Bain. *Learning Logical Exceptions in Chess*. PhD thesis, University of Strathclyde, 1994.

[BGR01] S. Babu, M. Garofalakis, and R. Rastogi. Spartan: A model-based semantic compression system for massive data tables, 2001.

[BMS95] M. Bain, S. H. Muggleton, and A. Srinivasan. Generalising closed world specialisation: A chess end game application, 1995.

[Bra01] I. Bratko. *Prolog Programming for Artificial Intelligence*. Addison-Wesley Publishing Company, 3rd edition, 2001.

[Edw95] S. J. Edwards. Comments on barth's article "combining knowledge and search to yield infaillible endgame programs.". *ICCA Journal*, 18(4):219–225, 1995.

[FPSM92] W. Frawley, G. Piatetsky-Shapiro, and C. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, pages 213–228, Fall 1992.

[Fue93] J. Fuernkranz. A numerical analysis of the KRK domain, April 1993.

[GL90] A. Guessoum and J. Lloyd. Updating knowledge bases. *New Generation Computing*, 8(1):71–89, 1990.

[GL91] A. Guessoum and J. Lloyd. Updating knowledge bases ii. *New Generation Computing*, 10(1):73–100, 1991.

[HMS01] D. J. Hand, H. Mannila, and P. Smyth. *Principles of data mining / David Hand, Heikki Mannila, Padhraic Smyth.* Cambridge, MA ; London : MIT Press, 2001.

[LaN14] *La Nature*, pages 56–61, 1914.

[Lev88] D. Levy. *Computer Chess Compendium.* B. T. Batsford, London, 1988.

[Llo87] J. Lloyd. *Logic Programming.* Springer-Verlag, Berlin, 2nd edition, 1987.

[MF92] S. Muggleton and C. Feng. Efficient induction of logic programs. In S. Muggleton, editor, *Inductive Logic Programming*, pages 281–298. Academic Press, London, 1992.

[Mit97] T. M. Mitchell. *Machine Learning.* The McGraw-Hill Companies, Inc., 1997.

[Mor92] E. Morales. *First order induction of patterns in chess.* PhD thesis, The Turing Institue - University of Strathclyde, 1992.

[Mor97] E. Morales. On learning how to play. In H. J. van den Herik and J. W. H. M. Uiterwijk, editors, *Advances in Computer Chess 8*, pages 235–250. Universiteit Maastricht, 1997.

[NWH99] E. V. Nalimov, C. Wirth, and J.M.C. Haworth. KQQKQQ and the Kasparov-World Game. *ICCA Journal*, 22(4):195–212, 1999.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Qui93] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[Roy86] J. Roycroft. Database "oracles": Necessary and desirable features. *ICCA Journal*, 8(2):100–104, 1986.

[Sad06] A. Sadikov. Machine learning. 2006.

[Tho86] K. Thompson. Retrograde analysis of certain endgames. *ICCA Journal*, 9(3):131–139, 1986.

[WF05] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, second edition, 2005.

[Wro93] S. Wrobel. On the proper definition of minimality in specialization and theory revision. In *ECML-93: Proc. of the European Conf. on Machine Learning*, 1993.

List of Figures

List of Tables