Machine Discovery Terminology

Willi Klösgen kloesgen@gmd.de
German National Research Center for Computer Science (GMD), 53757 Sankt Augustin, Germany

Jan Zytkow zytkow@wise.cs.twsu.edu
Department of Computer Science, Wichita State University, Wichita, KS 67208, U.S.A.

Introduction: We compiled this preliminary list of terms relevant for Machine Discovery, their definitions, and their most characteristic contents. The final goal is to describe the role of Machine Discovery and simplify the discussions within Machine Discovery Community. However, our current definitions are neither complete, nor adequate, while their sequencing and grouping of terms may not be satisfactory. We invite you to participate in the elaboration and refinement process. Therefore, comments and revisions to the definitions and their groups, and suggested additional terms are most welcome. Finally, any remarks to the implementation of this discussion process which shall be based on WWW are very welcome.

A definition may use terms which may require further definitions. To end the chain of definitions, some terms must be left undefined. We do not try to define terms which are technical in other disciplines. We also do not try to define common sense terms. Some of the key terms, which we left undefined, include "knowledge", "theory", "model". They have common sense meaning which we utilize, and they also have been technically defined in disciplines such as logic and philosophy of science.

Our definitions do not lead to increasingly abstract concepts. To the contrary, we frequently define by enumeration of examples. This makes definitions concrete and allows us to end the definition chains.

1. Discovery Systems

MACHINE DISCOVERY is a subfield of Artificial Intelligence which develops *discovery methods* and *discovery systems* to support *knowledge discovery processes*.

KNOWLEDGE DISCOVERY PROCESS aims at finding out *new knowledge* about an *application domain*. Typically, a discovery process consists of many *discovery steps*, each attempting at the completion of a particular *discovery task*, and accomplished by the application of a *discovery method*. The discovery process iterates many times through the same domain, typically based on *search* in various hypotheses spaces. *New knowledge* is inferred from data often with the use of old knowledge. *Domain exploration* and *discovery focussing* are discovery processes applied in new domains, where old knowledge is not available.

DISCOVERY STEP is a part of a discovery process. Discovery steps, *methods*, and *tasks* are interrelated: step is an application of a method; method accomplishes a task; task rationalizes a method. The main discovery steps include data collection, *pattern extraction from data*, *inductive generalizations* of data, *knowledge verification*, *knowledge transformation*. A knowledge discovery process may use steps which do not directly lead to new knowledge, but

enable further discoveries, such as *knowledge presentation*, management of *data*, management of *domain knowledge*, and selection of new goals.

DISCOVERY METHOD is an algorithm which performs a step of a *discovery process*. To have a theoretical value, the algorithm is supported by a theoretical framework. A discovery method can be a reconstructed human activity actually used to acquire *new knowledge*, can combine human methods in a novel way, but can be also a new method. Research areas from which *Machine Discovery* adapts pieces of discovery methods include Machine Learning, Statistics, Intelligent Database Management, Visualization, and *heuristic search* and *knowledge representation* in Artificial Intelligence.

DISCOVERY TASK is a direct or indirect specification of a component of new knowledge. Discovery tasks are open. They can be characterized by various spaces of possible solutions.

DISCOVERY SYSTEM is a software and possibly also hardware system that performs or supports a user in performing *knowledge discovery processes*. Typically, a discovery system integrates various *discovery methods*, and can be used in interactive and iterative ways. Discovery systems can be compared by evaluating their *autonomy* and *versatility*.

AUTONOMY measures to what extent a discovery system evaluates its decisions and produces *new knowledge* automatically, without external intervention. The degree of autonomy ranges from "apprentice systems" with low autonomy over "assistant systems" to "associate" and "master systems" which are nearly automatic discoverers.

VERSATILITY measures the variety of *application domains* and steps of *discovery processes* which a *discovery system* supports.

MAJOR DIRECTIONS IN MACHINE DISCOVERY are characterized by similar requirements and similar characteristics of the *discovery process*. The main directions are *Knowledge Discovery in Databases* and *Automated Scientific Discovery*. Future directions may include "Discovery in Mathematics", "Robotic Discoverers", and the like.

KNOWLEDGE DISCOVERY IN DATABASES (KDD) is a *major direction in machine discovery* dealing with *knowledge discovery processes* in databases. KDD applies to the ready data available in all domains of science and in applied domains of marketing, planning, controlling, etc. Typically, KDD has to deal with *inconclusive data*, *noisy data*, and *sparse data*.

AUTOMATED SCIENTIFIC DISCOVERY (ASD) is another *major direction in machine discovery* dealing with *knowledge discovery processes* analogous to those used by working scientists to make their discoveries. In distinction to KDD, a discovery process in ASD may seek additional data to improve the quality and expand the scope of generated knowledge. ASD covers fields of Natural Sciences (Astronomy, Biology, Chemistry, Physics, etc.), Medicine, and Social Sciences (Economy, Sociology, etc.).

2. World

APPLICATION DOMAIN is a real or abstract system existing independently from the *discovery system*. An application domain consists of *objects*, which can belong to one or several classes, and of object *attributes* and *relationships* between objects. Rather than to the whole world, *discovery systems* apply to limited application domains, with the intent to discover useful *domain models* and *domain theory*. In empirical discovery, the application domain becomes known by *data*, from which a *discovery process* attempts to generate *new knowledge*.

OBJECT (entity, unit, case) is a member or a part of an *application domain* (*universe*). Objects can belong to different classes of similar objects, such as persons, transactions, locations, events, and processes. Objects possess *attributes* and *relationships* to other objects.

ATTRIBUTE (field, variable) characterizes a single aspect of *objects* of an object class. An attribute has a value for each object in that class. This value is typically a number or a label. The value may be also a complex structure like a time series or even a picture that represents a person or a location in a multi-media application.

RELATIONSHIP (relation) combines *objects* from several object classes. A relation can be seen as a subset of a product set of several object classes. The relation holds for elements of this subset.

3. Knowledge

DOMAIN MODEL is a representation of one or several classes of *objects* of the *application domain* and some of their *relations*. The set of all objects forms the *universe* of the model. Domain model represents the perspective that a *discovery process* has on the application domain. A domain model can include *data* and *domain knowledge*. The formalisms used to build a domain model range from simple data files with added *data dictionary* to *knowledge representation* paradigms of Artificial Intelligence. An initially defined domain model is gradually elaborated in the course of knowledge discovery processes to achieve a *domain theory*.

UNIVERSE. For a class of objects represented in a *domain model*, a universe is the total set of all possible objects of this class. Often a probability measure is given or assumed for the universe. The subset of objects represented in the *domain model* belongs to a *sample set*.

SAMPLE SET is the subset of objects of an *universe* which are represented in the *domain model* and for which *data* are available. Often some probabilistic properties of the sample set are given or assumed.

DOMAIN THEORY is a comprehensive, consistent, and valid model of the *application domain*. Depending on the *knowledge representation* used to build the model, these characteristics measuring the quality of a domain theory are more formally defined.

DOMAIN KNOWLEDGE holds all information specific to the *application domain*, not belonging to *data*.

Some examples of domain knowledge ...

NEW KNOWLEDGE augments or refines the contents of the current *domain model*. New knowledge can be presented to the user and can extend the user's mental model of the *application domain*. The "user" of a *discovery system* can also be the same or another system, and new knowledge can augment the performance of this system, and may be used in making further discoveries. Typically, the user supervises the refinement of the domain model by new knowledge. New knowledge is described within a *knowledge representation* formalism.

KNOWLEDGE REPRESENTATION deals with data structures representing knowledge about many *application domains*. Artificial Intelligence offers knowledge representation paradigms like frames, production rules, semantic networks, first order logic. Typical knowledge representation structures used in *discovery systems* are *patterns* like *trees*, *rules*, *functional relations*.

4. Data

DATA consist of the collected (measured, sensed, polled, observed, etc.) *attribute* values for *objects* and *relationships* between objects in the *application domain*. Data coming from experiments include the results of manipulations and the subsequent readings of sensors. For the sake of completeness, special values *Missing Data* or *Not applicable data* can be used. Data can be arranged in various *data formats*. The meaning of data in databases is represented by a *data dictionary*. In *Automated Scientific Discovery*, the meaning of data is represented by manipulators and sensors and operating procedures through which they acquire data. The volume of data may be measured in bytes or the number of records times the number of attributes and in practical applications ranges from bytes to terabytes.

DATA FORMAT is a data structure to represent a particular piece of *data*. Data formats may be different for different applications. For instance, data about a particular object can be arranged into a *record*, and many records can be arranged into a *data matrix*. The *attribute type* describes the set of values of a given attribute and the meaningful operations on those values. *Discovery methods* may be limited to special *data types*.

INCONCLUSIVE DATA. Especially in *Knowledge Discovery in Databases (KDD)* applications, the available databases are installed for special purposes which may differ from the KDD purposes. Therefore, some *attributes* which may be relevant for a *discovery process* are often missing in *data*. Those hidden variables may be important and their absence may make it impossible to discover significant knowledge about a given domain.

NOISY DATA. Often *data* are infected with errors due to the nature of the collection, measuring, or sensoring procedures. Statistical methods can treat problems of noisy data.

MISSING DATA. *Attribute* values for some *objects* may be missing, because they were not measured, not answered, or simply lost. *Discovery methods* can treat missing data by omitting the corresponding *records*, inferring values for the missing values, or treat missing data as a special value to be included additionally in the *attribute domain*.

NOT APPLICABLE DATA. Sometimes *attribute* values are missing, because they are logically impossible for some *objects*, like the value "pregnant" for "male" objects. Information about this special kind of *missing data* can be included in the *domain knowledge* and can be treated in a special way by *discovery methods*.

SPARSE DATA. The events actually represented in given databases or *sample sets* typically build only a very small (sparse) subset of the *event space*. The order of magnitude is much higher for the *event space*, due to the abundance of combinations in building the product set of the *attribute domains*. Especially, this holds for *sample sets*.

EXTERNAL DATA refers to the permanently stored data and data structure. External data are often stored in a database management system. A *discovery system* can transform data available in a database system into its own special external data organization to speed up access and processing of data.

INTERNAL DATA refers to the data and its structure that is processed by a *discovery method* in main memory. Internal data are typically organized in *data matrices*. Discovery methods may process data incrementally. In this case, a loop over the input data can be organized, where at each step of the loop, only a small part of the input data is used by the method when processing the input data. A special incremental technique is *data driven search*.

RECORD is the collection of *data* belonging to one *object*. In the relational model, it is also called a tuple.

VIRTUAL ATTRIBUTE derives a value for each object of an *object* class by some user defined specification (transformation, method, etc.). Often, this specification refers to other *attributes*.

ATTRIBUTE TYPE of an *attribute* can be *nominal*, *ordinal*, *continuous*, *complex*.

ATTRIBUTE DOMAIN is the set of possible values of an *attribute*.

NOMINAL is an *attribute type* characterizing an *attribute* with an *attribute domain* for which no ordering is given.

ORDINAL is an *attribute type* characterizing an *attribute* with an *attribute domain* with an ordering.

CONTINUOUS is an *attribute type* characterizing an *attribute* with an *attribute domain* of a (dense) subset of real numbers. RATIO is a subtype with an interpretation of arithmetic operations (addition, multiplication) on domain values.

TAXONOMY is a hierachical system of subsets of an *attribute domain*, mostly arranged as a tree.

DATA MATRIX is a subset of *data* systematically organized into a matrix in which each row represents the values of all *attributes* (of a subset of attributes) for one *object* and each column represents values of one attribute for each object (of a subset of objects).

DATA TYPE characterizes (a subset of) *data* by the number of *object* classes and the *attribute types* of the *attributes* (that exist in this subset). Typical data types are *rectangular* and *multi relational*.

RECTANGULAR is a simple *data type* characterizing *data* with one class of *objects* and non complex *attribute types*. In the relational model, this assumes a single table.

MULTI RELATIONAL is a *data type* characterizing *data* for several classes of *objects* with non complex *attribute types*. *Relations* are available connecting the object classes.

TIME-SERIES is a *data type* for time series as logical data units. Relational, object oriented, or special time series databases can be used to store time series. One *attribute* represents different moments of time; the values of this attribute are ordered. Other attributes store information about co-inciding properties of *objects*.

COMPLEX-STRUCTURE is a *data type* characterizing *data* that do not belong to the *rectangular*, *multi relational*, or *time-series* type. Examples of complex-structured data are chemical, genetical, physical structures, image data, text and multimedia data.

DATA DICTIONARY includes information about the *attribute types* and values.

5. Sets of objects

EVENT SPACE refers to a selection of *attributes* of one *object* class. The event space is the product set of the *attribute domains*. To each event, a set of objects is associated. If a probability measure is defined for the *universe*, an event holds a probability.

CONCEPT is a subset of *objects* which may have some relevance in the *application domain*. Often a concept is defined by an event belonging to some *event space*, if a concept refers to one object class. In case of several object classes, a concept is a product set of subsets of object classes, defined by predicates. A *concept language* determines the concepts that can be defined.

CONCEPT LANGUAGE is used to construct *concepts*. Typical languages are *first order concept languages* and *propositional concept languages*.

CONCEPT SPACE is the set of all *concepts* which may be built within a *concept language*. The number of elements of this space depends on the type of the concept language. For languages of *strictly conjunctive form* of order n with no *internal disjunctions*, this number is mostly limited enough to prevent severe combinatorial problems. The problem of combinatorial explosion, however, is usually present for *disjunctive normal forms* without any order limitations. A concept lattice is given by the partially ordered space of concept extensions (subsets of objects with set inclusion as partial ordering) and the partially ordered space of concept descriptions (terms in the *concept language* partially ordered by generality).

FIRST ORDER CONCEPT LANGUAGES use some subset of predicate logic, mostly function-free Horn clauses, to represent *concepts* and *rule* patterns.

PROPOSITIONAL CONCEPT LANGUAGES (attributive languages) refer to conditions on *attributes* and their values. The main subtypes of these languages are *strictly conjunctive form* and *disjunctive normal form*.

STRICTLY CONJUNCTIVE FORM is a *(propositional) concept language* with terms built by conjunctions of *selectors*. A main subtype is the strictly conjunctive form of order n, allowing at most n conjunctions. Other subtypes are defined by restrictions for the construction of *selectors*.

SELECTOR defines a selection condition with an *attribute* and one or several values of the *attribute domain*. In case of an *ordinal* attribute type, one or several intervals may appear in a selector. An *internal disjunction* includes several values or intervals.

INTERNAL DISJUNCTION is a disjunctive selection built with several values of one *attribute domain*. In case of an *ordinal* attribute type, also a disjunction of several intervals is possible. To restrict the number of internal disjunctions, *taxonomies* can be defined.

DISJUNCTIVE NORMAL FORM is a *(propositional) concept language* with terms built by one or several disjunctions of conjunctions of *selectors*. The number of disjunctions is limited by n for disjunctive normal form of order n.

CONCEPT CLASSES are a set of *concepts*. *Rule* patterns refer to concept classes in their conditional or conclusion parts. Typically concept classes are disjoint. They form a partition, if they also cover all *objects*.

6. Patterns

PATTERN is a statement class which can be also regarded as a generic statement with free variables. Instantiated patterns (*pattern instance*) are candidates (hypotheses) to capture *new knowledge* on an *application domain*. An *evaluation* of a candidate exploits *data* and *domain knowledge*. A pattern is defined by a *pattern representation*. Various *pattern types* are applied in *Machine Discovery*.

PATTERN INSTANCE is a member of a *pattern* class. It can capture an elemental part of *new knowledge* like a single *rule* or a composite part like a system of rules or a *tree*. A pattern instance is fixed by an instantiation of the free variables in the generic statement belonging to the pattern class. A pattern instance is a statement *S* in a *pattern language* describing relationships among a subset *DS* of *data* of the *application domain* with *interestingness* *i*. *S* is simpler than an enumeration of all *records* in *DS*.

PATTERN LANGUAGE is a formalism to communicate *new knowledge* on an *application domain*. The kind of statements constructed in such a language depends on the *pattern type* and varies from natural-language-like sentences like *rules* to more abstract statements like *trees* or even graphical statements of a graphical language. An important component of a pattern language is the *concept language* used to build *concepts* within patterns.

PATTERN REPRESENTATION refers to the representation of a *pattern* in a *discovery system*. The main representation components refer to *pattern extraction*, *evaluation*, *presentation specifications*, and *pattern arguments*.

PATTERN EXTRACTION is a major *discovery task*. For the various *patterns* and *pattern types*, special pattern extraction methods (e.g. *tree extraction method*, *rule extraction method*, *functional dependency extraction method*, or *statistical pattern extraction method*) discover *new knowledge* in the form of *pattern instances*. Pattern extraction methods rely on *search* and *evaluation*.

EVALUATION checks a *pattern instance* by measuring its *interestingness*. An *application test* can verify some preconditions for the interestingness of an instance. In case of a composite instance like a *tree*, a system of *rules*, or an *equation*, also the components of the instance (node, single rule, term of an equation) can be evaluated.

PRESENTATION SPECIFICATIONS determine, how the contents of a *pattern instance* are presented to the user, e.g. in natural language, tabular, graphical, or audio-visual form. *Presentation templates* are typical simple presentation specifications.

PRESENTATION TEMPLATE is a schema for a textual or graphical presentation of a statement (*pattern instance*). Typically such a schema has some parameters. The values of the parameters are fixed by the pattern instance.

PATTERN ARGUMENTS correspond to free variables in the generic statement of a *pattern*. This component includes specifications on the admissible instantiations of the free variables and their properties, e.g. *extraction properties* exploited by a *pattern extraction* method. *Range* is an argument which is available in most patterns.

RANGE is a subset of *objects*. Typically it is defined by a logical condition on some *attributes* and their values (*concept language*). It is used to restrict the scope of a *pattern* to a subset of *objects*. If the statement refers to several object classes, a range is a product of subsets of objects of these classes.

EXTRACTION PROPERTIES of *pattern arguments* are exploited by a *pattern extraction* method to construct a *search space* and operate on it. E.g., extraction properties can determine conditions, that exclude all subnodes of a node from further *search*.

EXTRACTION GOALS are general directives for *pattern extraction* specified by the user of a *discovery system* during *discovery focussing* . They relate to the application purpose of the *new knowledge* to be discovered (e.g. accurate classification or structure uncovering), *pattern language*, *evaluation*, and extraction effort (granularity and extent of *search*).

INTERESTINGNESS of a *pattern instance* measures its quality and has several dimensions. The main dimensions are the *validation* on the *sample set*, the *reliability* on the *universe*, the degree of *redundancy* with respect to other already known pattern instances, the *generality*, the *simplicity*, and the *usefulness*.

VALIDATION checks a *pattern instance* (or component of an instance) referring to the subset of *data* in the *sample set* which is connected with the instance. To check an instance, usually a statistical test or some other criteria are validated. Additionally to the decision, whether an instance is valid or not, often also an *evidence* measure is calculated.

EVIDENCE measures the statistical significance or some other kind of conspicuousness of a *pattern instance*.

APPLICATION TEST is a filter which includes some preconditions for a *pattern instance* to be evaluated as *interesting*. The filter is used to avoid possibly extensive evaluation efforts, when the interestingness of an instance can be excluded already by the preconditions.

RELIABILITY includes some estimation on the validity in the *universe* of the *pattern instance* which was discovered in the *sample set*. Cross-validation methods can be applied to derive some estimation on the correctness of the statement (pattern instance) in the universe.

REDUNDANCY relates to several *pattern instances* or to several knowledge components of a complex pattern instance (e.g. nodes in a *tree*). Redundancy is given, if one instance or component follows (logically) from another one. Quantification of redundancy can be introduced to measure the conditional probability of one instance or component given the other one.

GENERALITY measures the strength of a *pattern instance* in terms of the size of the subset of objects which are described by the statement.

SIMPLICITY measures the syntactical complexity of a statement (*pattern instance*).

USEFULNESS quantifies the possible usefulness of a statement (*pattern instance*). The usefulness can be related to a task the user of a *discovery system* has to perform or to a task that a computer system can perform on the basis on the discovered *new knowledge*.

PATTERN TYPES are classes of *patterns*. The main classes are *logic-numerical pattern*, *elementary pattern*, and *complex pattern*.

LOGIC-NUMERICAL PATTERN holds the subclasses *tree*, *rule*, *functional relation*, *statistical pattern*.

ELEMENTARY PATTERNS work on aggregations of *data*. Typical aggregations are given by reports (e.g. based on group-by SQL operations) or multi-dimensional tabulations. Aggregation operations include count, sum, max, min, average, etc. Elementary patterns do not involve a complex *search* process and investigate rows or columns of these tabulations e.g. for

monotony, convexity, concavity, maximum, minimum, discontinuity, outlier. Several rows or columns can also be compared (e.g. all cells in one row are larger than the corresponding cells in another row).

COMPLEX PATTERNS are patterns not of the *elementary pattern* or *logic-numerical pattern* type. They are relevant for discovery in *application domains* with *data* of the type *complex structure*.

TREE is a tree-like partition of an *universe* or *sample set* into a hierachically ordered set of *concepts*. Each concept on a hierarchical level is recursively divided into subconcepts on a next lower hierarchical level. Typically, concepts on each hierarchical level are disjoint and collectively exhaustive, and the description of the subconcepts on the next level (*concept language*) includes a further conjunctive term built with one further *attribute*. The main subtypes of this *pattern type* are *classification trees* and *regression trees*.

TREE EXTRACTION METHOD uses criteria to select a (next) *attribute* for each *concept* on a hierarchical level, to divide the *attribute domain* of this attribute in (disjoint) subsets which correspond to the subconcepts on the next level, to terminate further partitioning of a concept, and to *prune* the tree. The criteria used by the extraction method depend on the *extraction goals*.

CLASSIFICATION TREE is a *tree* representing a set of *classification rules* for *concept classes*. Each leave of a classification tree is associated to a concept class, where the description of the leave constitutes a sufficient condition for the concept class. Classification trees can be used to classify objects following the concept descriptions from root to leaves.

REGRESSION TREE is a *tree* representing a set of homogenous *concepts*. A concept (node) in this tree is homogeneous referring to a *continuous attribute*, i.e. the variance of this attribute in the concept is minimal.

RULE correlates two *concepts*. Typically, the left hand side concept LHS is a sufficient condition for the right hand side concept RHS. A rule can be presented as: If LHS then RHS. There are *exact*, *strong*, and *probabilistic rules* as well as *attributive* and *first order rules*.

EXACT RULE allows no exceptions. Each object of the LHS *concept* of a *rule* must also be an element of the RHS concept.

STRONG RULE allows some exceptions. The number of exceptions mostly may not exceed a given limit expressed as percentage or absolutely.

PROBABILISTIC RULE relates the conditional probability P(RHS | LHS) to the probability P(RHS).

ATTRIBUTIVE or PROPOSITIONAL RULE is based on a *propositional concept language*.

FIRST ORDER RULE is based on a *first order concept language*. In this case, LHS and RHS concepts are sets of object tuples (including several object classes).

CLASSIFICATION RULE belongs to a system of classification *rules* which has to be discovered for given right hand side *concept classes*.

CHARACTERISTIC RULE belongs to a system of characteristic *rules* which has to be discovered for a given left hand side *concept*.

RULE EXTRACTION METHOD discovers a system of *rules*. The *extraction goals* include the desired subtypes of rules, (partly) fixing LHS resp. RHS *concepts*, and other goals like predictivity (maximal classification accuracy) or uncovering of structure.

FUNCTIONAL RELATION is a *pattern* relating a dependent *attribute* to one or several independent attributes. *Functional dependency* and *equation* are subtypes.

FUNCTIONAL DEPENDENCY exists between a dependent *attribute* and some independent attributes, if for each pair of objects with equal values of the independent attributes the values of the dependent attribute are equal too. An APPROXIMATE FUNCTIONAL DEPENDENCY allows some exceptions (e.g. due to noise).

EQUATION is a pattern which relates a dependent *attribute* to independent attributes in the form of a mathematical functional equation.

FUNCTIONAL RELATIONSHIP EXTRACTION METHOD discovers the existence and/or equation of a functional relationship. Typically *search spaces* of terms connected by mathematical operations are generated and processed to identify equations. In some *application domains*, it is useful to construct the space of terms from predefined component terms.

STATISTICAL PATTERN describes significant *concepts*. The significancy of a concept is verified by a statistical test based on a hypothesis on the concept. *Statistical dependency patterns* are a main subtype.

STATISTICAL DEPENDENCY PATTERNS are *statistical patterns*, for which the hypothesis on a *concept* relates to the distribution of a dependent *attribute* in the concept. Subtypes of this *pattern* are given by distinguishing the *attribute type* of the dependent *attribute* and some distribution parameters. The concept (resp. the distribution) can be related to the whole *range*. The concept can also be compared e.g. for different time points.

STATISTICAL PATTERN EXTRACTION METHOD operates on a *search space* of *concepts*.

7. Search

DISCOVERY FOCUSSING is a major *discovery task* which has the aim to fix an individual discovery problem. The main specifications resulting from this task relate to selecting a subset of *data* to be analysed and to *extraction goals*. Discovery Focussing is primarily done by the user of a *discovery system*. However, a system can transform more global and nontechnical specifications of the user into its technical constructs.

SEARCH is the central approach for *pattern extraction*. Search is performed in a *search space* usually by exploiting some structure in this space. Different *search strategies* can be applied by pattern extraction methods to generate and process the search space. Search can be arranged in several search phases or iterations. *Search refinement* can be provided to refine the results of a preliminary search.

SEARCH SPACE is a one or multi-dimensional space with a partial ordering. The elements (nodes) of a search space can correspond to *pattern instances* (e.g. *rules*) or to components of pattern instances (e.g. conjunct within a *rule* or single rule within a system of rules, node of a *tree*, or term of an *equation*). Search spaces have to be constructed by a *pattern extraction* method according to *discovery focussing*. Search spaces can be constructed statically before search or dynamically during search.

SEARCH STRATEGIES are general approaches to construct and process *search spaces*. The main strategies are *heuristic search*, *exhaustive search*, *data driven search*, and *concept driven search*.

HEURISTIC SEARCH is a *search strategy* applied to generate and/or process only a part of a total *search space* which includes all possible *pattern instances* or components of pattern instances. Heuristic criteria determine, which parts are included into search. Typically heuristic search generates a satisfying solution, but not an optimal solution. Often search spaces are so large, that only heuristic search can produce a solution in reasonable time. *One step optimal search* and *beam search* belong to the main heuristic search approaches.

ONE STEP OPTIMAL SEARCH (or stepwise search) is a *heuristic search* strategy that is performed in several recursive steps. At each step, successor node(s) of a node are determined which optimize a given local criterium.

BEAM SEARCH is a *heuristic search* strategy similar to *one step optimal search*. At each step, the best n partial solutions are determined according to the local optimizing criterium and further processed.

EXHAUSTIVE SEARCH processes and evaluates all nodes of a *search space*, possibly omitting those nodes which can be excluded as not *interesting*. Exhaustive search ensures an optimal solution, but often is not realistic because of time constraints.

DATA DRIVEN SEARCH organizes the major loop during *search* over the *records* of *data*. Each record is accessed sequentially and associated to a node in the *search space*. This node is updated according to the record. Several passes on *data* may proceed. After each path, some filter operation selects the best nodes in the search space according to some criterium and elaborates these nodes in the next pass. Data driven search minimizes *data* accesses and can result in time efficient discovery.

CONCEPT DRIVEN SEARCH organizes the major loop during *search* around the structure (e.g. partial ordering) of the *search space*. When a node of the search space is processed, the associated subset of *data* is accessed. If these accesses are performed randomly to *external data*, time efficiency of discovery may be a problem.

SEARCH REFINEMENT refines the results of a previous *search* phase. E.g., search granularity can be increased to search in the neighbourhood of a previously identified node. *Pruning* is another refinement technique.

PRUNING cuts search spaces. This can be done after search (postpruning) or during search. E.g., a *tree* can be cut, to eliminate overspecializations.

DOMAIN EXPLORATION is a major *discovery task*. ...